NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

EFFECTS OF DIFFERENT CAMERA MOTIONS ON THE ERROR IN ESTIMATES OF EPIPOLAR GEOMETRY BETWEEN TWO DIMENSIONAL IMAGES IN ORDER TO PROVIDE A FRAMEWORK FOR SOLUTIONS TO VISION BASED SIMULTANEOUS LOCALIZATION AND MAPPING (SLAM)

by

Michael Charles McVicker

September 2007

Thesis Co-Advisors: Mathias Kölsch
Kevin Squire

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE		*Form Approved OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2007	**3. REPORT TYPE AND DATES COVERED** Master's Thesis
4. TITLE AND SUBTITLE Effects of Different Camera Motions on the Error in Estimates of Epipolar Geometry between Two Dimensional Images in Order to Provide a Framework for Solutions to Vision Based Simultaneous Localization and Mapping (SLAM)		**5. FUNDING NUMBERS**
6. AUTHOR(S) Michael Charles McVicker		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

This thesis explores the effect camera motion and feature tracking have on the estimations of an epipolar geometry at different stages of a 3D reconstruction and relates the findings to a framework for vision based Simultaneous Localization and Mapping (SLAM). Although there have been previous attempts to determine the quality of algorithms that calculate a fundamental matrix, both robust and linear, we have found no study that explores the relationship between camera motion, or likewise the different types of parallax, and errors in the epipolar geometry between two images as defined by an estimated fundamental matrix. The interest comes from the fact that there are claims to this end made by two prominent textbooks in this area. By using synthetic scenes that are projected with and without noise by camera matrices that define different camera motions between the projections we are able to isolate the three different type of parallax that can be experienced between projections; no parallax shift from rotational movement, a high amount of parallax shift from translational movement in the camera's xy-plane, a high amount of parallax shift from translational movement along the camera's optical axis (z-plane). We also studied an unconstrained movement with components of each of the previous three types. The different camera motions are equivalent to different motions a robot would experience when performing SLAM, specifically, rotational, lateral, forward and unconstrained motions. There are multiple experiments that explore the effect motion has at every stage of a projective reconstruction algorithm.

14. SUBJECT TERMS Parallax Shift, Parallax, Epipolar Geometry, Fundamental Matrix, Camera Motion, 3D Reconstruction, Vision SLAM, Simultaneous Localization and Mapping.	**15. NUMBER OF PAGES** 195
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

EFFECTS OF DIFFERENT CAMERA MOTIONS ON THE ERROR IN ESTIMATES OF EPIPOLAR GEOMETRY BETWEEN TWO DIMENSIONAL IMAGES IN ORDER TO PROVIDE A FRAMEWORK FOR SOLUTIONS TO VISION BASED SIMULTANEOUS LOCALIZATION AND MAPPING (SLAM)

Michael C. McVicker
Captain, United States Marine Corps
B.A., University of Michigan - Ann Arbor, 1997

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2007

Author: Michael Charles McVicker

Approved by: Dr. Mathias Kölsch
Thesis Co-Advisor

Dr. Kevin Squire
Thesis Co-Advisor

Dr. Peter Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis explores the effect camera motion and feature tracking have on the estimations of an epipolar geometry at different stages of a 3D reconstruction and relates the findings to a framework for vision based Simultaneous Localization and Mapping (SLAM). Although there have been previous attempts to determine the quality of algorithms that calculate a fundamental matrix, both robust and linear, we have found no study that explores the relationship between camera motion, or likewise the different types of parallax, and errors in the epipolar geometry between two images as defined by an estimated fundamental matrix. The interest comes from the fact that there are claims to this end made by two prominent textbooks in this area. By using synthetic scenes that are projected with and without noise by camera matrices that define different camera motions between the projections we are able to isolate the three different type of parallax that can be experienced between projections; no parallax shift from rotational movement, a high amount of parallax shift from translational movement in the camera's *xy*-plane, a high amount of parallax shift from translational movement along the camera's optical axis (*z*-plane). We also studied an unconstrained movement with components of each of the previous three types. The different camera motions are equivalent to different motions a robot would experience when performing SLAM, specifically, rotational, lateral, forward and unconstrained motions. There are multiple experiments that explore the effect motion has at every stage of a projective reconstruction algorithm.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

First and foremost, I want to acknowledge God and his benevolence for giving me the strength and tenacity to actually accomplish this daunting task.

Second, and none the less, I thank my lovely wife and beautiful children for giving me the time and supporting me through this endeavor of both this thesis and the pursuit of a Masters in general.

I owe a great debt to both my thesis advisors for standing by me as I meandered through this abstract concept of research. The countless emails and requests of their time and effort were always appreciated, although perhaps never acknowledged until now.

I thank the Marine Corps, and specifically those that made my attendance here at this institution of higher learning possible. This has been a life-altering experience that will forever impact me and my family.

Lastly, I would like to thank the members and directors of the Autonomous Systems Lab for allowing me to bivouac in the lab while writing this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Imagine you are an infantry platoon commander on the streets of a hostile city. Your orders are to clear the buildings in your area of operations of all hostile threats. You understand that historically this is one of the most dangerous jobs in the military. Luckily, you have something novel. You have a cluster of tiny, autonomous, micro-flying robots in your cargo pocket. Prior to your entry, you pull these robots out and send them in first. The size and cost of these robots severely limits their capabilities. They can only navigate autonomously, and wirelessly transmit captured images to a handheld device. Their small camera is similar to that of the webcams ubiquitous throughout America today. As the robots enter the building through a window you look down on the screen of the small handheld computer. Immediately you begin seeing a three-dimensional virtual model of the building appear and you begin virtually touring the building as the robots progress through it. Within minutes, you have a complete virtual three dimensional map of the entire building and you now know where people are located, where they could be hiding, and what traps await you. You quickly brief your platoon, enter the building, and successfully clear the building suffering no casualties.

Of course the story is fictitious because we neither have flying 'bugs' in our arsenal nor the handheld computers that they communicate with; however, the fact is that the technology to create such a device exists today. There are various flying micro-robots that have been developed with both cameras and wireless transmitters. For example, Seiko Epson, Corp. produced the micro flying robot in Figure 1.

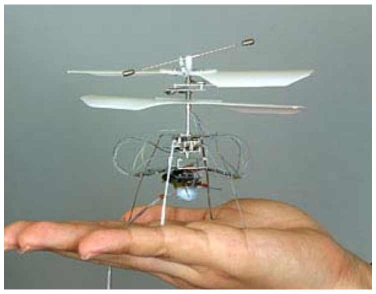


Figure 1. Micro-flying robot created by Seiko Epson, Corp.
From: http://www.epson.co.jp/e/newsroom/news_2004_08_18.htm

The only thing lacking is a suitable real-time system that provides reliable navigation using a monocular camera that can communicate with another system that simultaneously creates a virtual model of the environment from multiple robots. This issue is far easier said than done, and its solution lies in a combination of hardware improvements and software algorithms. We propose that the framework for a software algorithm can be built from the coupling of two different fields in computer science: robotics, and computer vision.

In the hypothetical scenario the robot's only sensor that allows it to sense the outside world is its monocular camera. It may, just as easily, have a stereo camera rig. Whichever the case, the images it captures are its only connection to the world around it. In the robot's computer memory the images are merely a collection of 0's and 1's that needs to be processed into something more abstract that the robot can actually use. The processing of these images falls under the scope of researchers in the field of computer vision. On the other hand, for a robot to navigate, it requires the ability to map its surroundings, and simultaneously locate itself within the map. If a map were known, then the robots location is recovered from correlating sensor information with the known map. If its location were known, then mapping is done by simply filling in a 2-D or 3-D occupancy grid with information from its sensors. When neither a map nor the robots' location are known a priori, then there is a quandary between mapping and localizing. This occurs from the fact that any movement of the robot introduces significant noise in its location which results in near failure in its ability to map. To solve this quandary the robotics community offers algorithms under the general term SLAM (Simultaneous Localization and Mapping). These algorithms provide solutions using probability models and filters, such as Kalman, Extended Kalman, and particle filters. The dilemma for these approaches as it relates to the scenario is that they are focused on different active range sensors, such as infrared and LIDAR, which retrieves the three-dimensional data from the robot's surroundings directly. The exact opposite is true for the scenario's robots. The images the robot retrieves loses the depth information when it is transformed, projected, from three- to two-dimensions. However, if the third dimension can be recovered through image processing, then this information can be used in a similar manner as the data provided by range sensors.

But why cameras? Why not just use range sensors? The range sensors are considered active because they determine depth by emitting a signal and interpreting the response. This ability comes at a cost of a large payload with respect to its weight and energy requirements, hence it is too much for such a small robot. A camera, on the other hand, is passive and merely needs to measure light that is naturally reflected off objects. The associated payload with passive sensors is smaller which in turn is why many of the micro-flying robots being developed, like our example, include cameras.

Another noticeable difference between active and passive sensors is their ability to see in adverse conditions. An active sensor such as the SICK laser has the ability to see through smoke and darkness because it does not rely on ambient light. A camera that operates in the visual light range, on the other hand, is effected in similar ways to how a human's ability is diminished in smoke, and darkness. This can be partially overcome through sensor fusion. For instance, if there were two collocated cameras, one operating in the visual light spectrum, the other operating in the infrared spectrum. The infrared could see through smoke, but may not give the information necessary to perform a 3D reconstruction as described herein.

In order to retrieve the third dimension from two-dimensional images, the computer vision community offers algorithms referred to as stereo vision and Structure from Motion (SFM) (also known as Structure and Motion (SAM)). In general, stereo vision requires the use of two cameras that have been calibrated to work together. We will focus on algorithms that primarily support systems using a monocular camera. SFM relies on the discovery and tracking of features from one image to another and requires that there be movement of the camera between image captures. The third dimension is recovered from these tracked features through estimation of the epipolar geometry between the images and then triangulation using the estimated camera poses and the tracked features. An explanation and further details of this process will be given in Chapter II and III.

Systems that include both SLAM algorithms and SFM algorithms may provide the technology necessary to robots with monocular camera to navigate using a three-dimensional sparse map created on the fly. The actual implementation of such a combined system has not been solved, although attempts have been made and the most notable are outlined in the literature review in Chapter III. In essence, the ability of the various algorithms from both SFM and SLAM to work together to provide accurate metric localization and mapping have yet to be fully understood and tested. Furthermore, there could be a lack of suitable hardware necessary if the only true way this goal can be accomplished is through the reproduction of biological vision (e.g., human vision capabilities); but this is beyond the scope of this thesis.

In order to begin evaluating the different algorithms' capabilities we must first determine the quality of SFM algorithms under different conditions. The algorithms' abilities to recover three-dimensional depth is directly related to both the quality of the tracked features between frames and the camera motion. The features are the data captured by the sensors, which are input into the algorithms. The motion of the robot directly influences how these tracked features move in relation to each other. Hence, for a monocular camera system, different camera motions coupled with various quality of features produces results for algorithms that are decidedly different.

A. MOTIVATION AND BENEFITS

We would like to create a system for robots to conduct SLAM using only vision. We assume that in order to perform SLAM a robot needs to relatively accurately sense its surroundings and move. Typically the sensing is done using an active range sensor such as a laser, but for our purposes we restrict the sensor to a single monocular camera. This forces us to determine a means of acquiring depth from 2D images. It has been shown that Structure from Motion (SFM) algorithms can reproduce a rigid 3D scene from 2D images using epipolar geometry which would give us the range information we need to sense our surroundings.

With regard to epipolar geometry, we know that studies have been done to ascertain the quality of estimations of the epipolar geometry by different approaches., and that there are degeneracies that can develop under different camera motions that effect the quality and ability to calculate an epipolar geometry between two projections. When coupling these we find that the different studies have not looked at the quality of calculated epipolar geometries under the different known degenerative conditions caused by camera motions. The reason these conditions arise is that there is a difference in levels of parallax shifts between points under different camera motions (explained in more detail in Chapter II). However, a robot with unconstrained motion will experience all possible camera motions, so we require further understanding of how algorithms perform under degenerative conditions and how to overcome those conditions. Furthermore, one advantage of a system on a robot is that it can not only detect motion from images, but it has an expectation of what that motion should be based upon the motion it is attempting to carry out. If a certain motion is expected and there is a higher quality algorithm for that particular motion, then the system can be optimized with respect to expected camera motion.

If there are camera motions that cause degenerative conditions that cannot be overcome by an algorithm, then that algorithm should not be considered for use in a system for conducting SLAM. Likewise, if there is a particular camera motion that frequently fails for an otherwise well performing algorithm, then the design of the robot and/or movement of the robot should be controlled to prevent this camera motion.

As a side consequence of our studies we happened upon a novel method for tracking features which is explored in experiment one. Although not directly related to the error caused by various camera motions, the higher quality the tracking of features between images the less need for a robust algorithm, hence our multi-pass feature tracking algorithm is a supplement to more well-known robust methods. This is explained in more detail in Experiment One.

B. SCOPE OF THIS THESIS

This thesis does not attempt to create a new algorithm for estimating epipolar geometry, but rather focuses on how well current algorithms perform under different camera motions. Furthermore, this thesis does not attempt to conduct SLAM or develop a system to do so. Instead the focus is on how accurate the available algorithms perform which in turn will provide the SLAM system with the best possible data to conduct SLAM.

Furthermore, there are certain assumptions made by these types of algorithms that we do not explore beyond. Namely, we do not explore the use of these algorithms on dynamic scenes. A static scene is where the subjects within the scene remain rigid between projections. Conversely, a dynamic scene is where there can be one or more subjects in the 3D scene which do not remain rigid between projections. Hence these cause noise in most SFM and SLAM algorithms which assume rigidity of the subjects. Possible ways to overcome this is to assume rigidity locally and dynamic subjects over the entire image stream, or perhaps include a background subtraction algorithm which can remove the foreground prior to, or during feature tracking. Again, this is beyond the scope of this thesis.

C. CHAPTER ORGANIZATION

The rest of this thesis is organized as follows. Chapter II provides the reader with an introduction to the theory behind computer vision and SFM algorithms. To the reader that is unfamiliar with these fields, it provides the terminology necessary to understand the issues and where our experimentation comes from.

Chapter III looks at recent work done in the area of 3D reconstruction and third-dimension recovery from 2D images. We claim that there is difference between the two based upon the method used. The difference is that a 3D reconstruction is more precise approach to recovering a third dimension that includes optimization and precision in measurements. Conversely, third-dimension recovery is done through imprecise where there is not any precision in its measurements. For example, recovering the third dimension of an object in a scene by way of occlusion from a dynamic subject in the scene (as seen in Chapter III). Although this method can provide details about the 3D nature of an object in a scene, there is no precise measurements being made and without any scene knowledge, there is no apparent way to find metric dimensions of the object or the scene itself, let alone a range image that is useable in SLAM.

Chapter IV describes the common aspects of the experiments to include the synthetic 3D scenes used in the projections and a definition of each of the camera motions.

The basics of the experiment assumes the knowledge from Chapter II and explains the design of each experiment. In essence, the experiments focus on the quality of the estimated epipolar geometry for different algorithms and camera motions using the same corresponding points as input.

Chapters V through X are the actual experiments. Each experiment begins with a description and motivation for the experiment, followed by the method for which the experiment is performed, and then a description of the results and a discussion of such.

The final chapter of the thesis, Chapter XI, gives a general summary of the thesis and experiment results as well as the opportunities for future work and conclusions.

We also include an extensive appendix (Chapters XII through XV containing detailed tabled results for experiments two through four. Each of these chapters includes a brief description at its beginning describing how to read the tables.

THIS PAGE INTENTIONALLY LEFT BLANK

II. 3-D RECONSTRUCTION PRIMER

For continued reading in this thesis there are certain concepts that should be understood. This chapter lays out the details of these concepts and can be used as a primer in 3D reconstruction, or, alternatively, as a reference when reading the experiments. Some of the topics are introduced as background information for terminology, where others are introduced so that the reader can better understand why the noise that plagues 3D reconstruction algorithms exists.

A. GEOMETRY

A fellow took a morning stroll. He first walked 10 mi South, then 10 mi West, and then 10 mi North. It so happened that he found himself back at his home. How can this be?

— Alexander Bogomolny
http://www.cut-the-knot.org/triangle/pythpar/NonEuclid.shtml

The above problem does indeed have an answer in that the fellow does end up back home; however, this problem does not lend itself to what is generally known about Euclidean geometry. In a Euclidean geometry, in order to traverse an area making 90 degree turns in a single direction and return to the starting point, one has to have trips that are equal distance, and turns that total 360 degrees if traveling in a plane. However, the solution to this problem falls from the fact that the fellow on the stroll is not on a Euclidean plane. Rather he is performing his stroll in the context of a non-Euclidean framework where it is indeed possible to return to an origin via 180 degrees of turns. The answer is that he is walking on a sphere and starts at a point, for example the North Pole, walks south any amount of distance, followed by a 90 degree turn west. He then proceeds west for any amount of distance and can return home at any moment by a ninety degree turn towards the north and walking an equal distance to how far he initially traveled south. This anecdote lays the foundation for our discussion on geometry.

Euclid lived in Alexandria around 300 B.C.E. and was the first to publish a book describing geometry called “Elements.” What Euclid introduced is now referred to as Euclidean geometry but for almost 2000 years after Euclid, this was the only geometry known. For our purposes, there are four different types of geometries: Euclidean, Metric, Affine, and Projective. The geometries are hierarchical from projective to Euclidean, meaning each subsequent geometry's invariant properties are inclusive of the geometries that are its lesser. For example, Affine geometry has the invariant properties of a projective reconstruction as well as the additional invariant properties of volume ratios, etc. The hierarchy is shown below in Table 1.

.

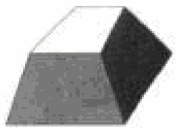
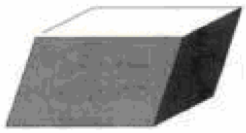
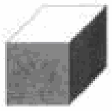
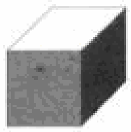
Group	Matrix	Distortion	Invariant properties
Projective 15 dof	$\begin{bmatrix} \mathrm{A} & \mathbf{t} \\ \mathbf{v}^{\mathsf{T}} & v \end{bmatrix}$		Intersection and tangency of surfaces in contact. Sign of Gaussian curvature.
Affine 12 dof	$\begin{bmatrix} \mathrm{A} & \mathbf{t} \\ \mathbf{0}^{\mathsf{T}} & 1 \end{bmatrix}$		Parallelism of planes, volume ratios, centroids. The plane at infinity, π_∞.
Euclidean 7 dof	$\begin{bmatrix} s\mathrm{R} & \mathbf{t} \\ \mathbf{0}^{\mathsf{T}} & 1 \end{bmatrix}$		The absolute conic, Ω_∞.
Metric 7 dof	$\begin{bmatrix} \mathrm{R} & \mathbf{t} \\ \mathbf{0}^{\mathsf{T}} & 1 \end{bmatrix}$		Volume.

Table 1. Invariant Properties of Different Geometries. From: [1]

Euclidean geometry is what most of us are familiar with when it comes to mathematics. It is the study of points, lines, planes and angles. It provides the geometry that necessitates walking in a square as a means of getting back to where you started.

Metric Geometry is a similar form of geometry to Euclidean; however, the scale of metric geometry is equivalent to the real world, where Euclidean can be of any scale. For example, there is no difference between a doll house and a real house in a 3D Euclidean reconstruction, but a metric reconstruction has the actual scale of the house. Hence a doll house can be distinguished volumetrically from a real house.

Leveraging knowledge of metric and Euclidean geometry as described above we would like a method to transform any other geometry to at least Euclidean, if not a metric geometry. This transformation is referred to as upgrading the geometry to its Euclidean/metric equivalent. To upgrade a Euclidean geometry to its metric equivalent, we need to know a metric measurement in the image. For instance, if we knew that a unit of measurement in our Euclidean geometry is equivalent to 20 feet in a metric geometry, then we can adjust all the units by multiplying by 20. To upgrade other geometries, we need to understand them and how they relate to a Euclidean geometry.

To understand affine geometry, you must first understand a projective geometry and its relation to, and what is meant by, a point at infinity and the plane at infinity (Π_∞). First, lines in Euclidean space that are parallel remain parallel forever and never intersect. In a projective geometry and likewise a perspective image in which a projective geometry is based upon, the lines do intersect due to the perspective of the observer is illustrated in Figure 2.

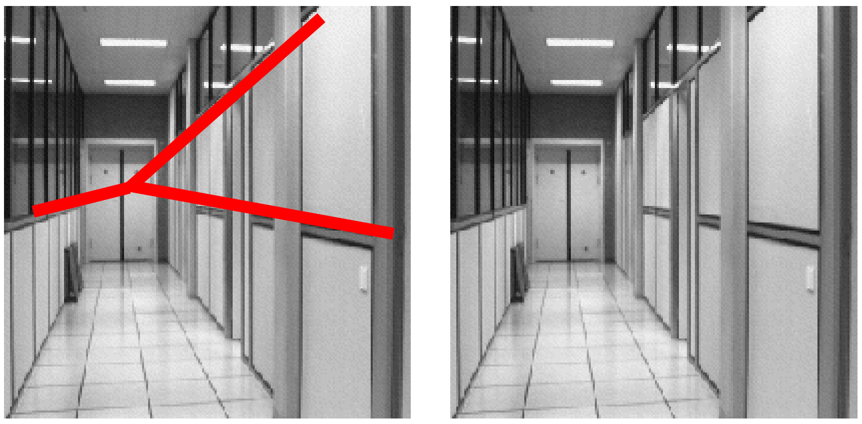


Figure 2. Point at infinity.

The above image (shown twice) shows how parallel lines in a perspective projection converge at a point at infinity. From:[1]

The point in which these lines intersect defines a point at infinity. It represents the point where the lines would converge if allowed to go on forever from the current perspective. If there are multiple sets of different parallel lines in an image, then we can find multiple points at infinity. At least three distinguishable points at infinity are needed to define the plane at infinity (Π_∞). Π_∞ is invariant in an Affine geometry, so is necessary when upgrading a projective reconstruction to an affine reconstruction. [2]

B. CAMERA SENSOR (CCD)

When an algorithm attempts to track features between images, there is error that exists in feature's (x,y) coordinate location. This noise has an effect on the resulting estimates in the epipolar geometry. The Cartesian coordinate contains noise due to the conversion of a feature from the analog world, to the digital image. This is explained below.

Every object in the real world reflects light waves of various wavelengths. When we "see" something, or rather visually perceive something, what is occurring is that reflected light waves are being focused by the eye's lens onto the retina. There, photoreceptive cells transform the analog light wave signals using the discrete number of photons into neural impulses that can be interpreted by the visual cortex of the brain. [3]

In order to bring images into the computer a similar process occurs. Lenses in a camera focus the light waves onto receptors that convert the analog signal into a digital signal that the computers can then perceive. A common example is a charged-couple device (CCD). This is an array of pixels made of silicon. When photons hit the silicon, the silicon releases electrons. The number of electrons released is linearly dependent on the amount of light that disturbs the silicon, i.e., on the number of photons that hit the silicon including noise from spillage of other ambient lightwaves. The number of electrons released by the silicon has no bearing on the wavelength (color) of the light effecting it. The result is a monochrome image. In order to extract color using this method, filters are used such as an Integral Color Filter Array (CFA). This system of filtering light collocates three different sensors at each pixel location with a different filter on each (red, green and blue). The filter determines the light waves that can pass through, so the sensors will only release electrons when their respective ranges of light waves are present. In turn the color of the pixel is determined by a combination of how excited each of the sensors gets. The end result is a bitmap, or image stored as ones and zeros in a computer. [4]

Due to the fact that the sensors are discretely localized and a light wave most likely will not land directly on these sensors, there is error between the true location and the actual measurable location of a pixel. This can be interpreted as noise when attempting to localize the corresponding points between images. Therefore, even before processing images, there is error in the actual pixel locations. To minimize this error, a larger sensor array can be used, but the costs are that the images generated are larger which increases the processing time for each image, the monetary cost of the sensor itself, and possibly more noise by way of smaller elements being detected in the image.

C. CAMERA CALIBRATION

A camera's lens also introduces noise into the Cartesian coordinate of features. A lens's role is to redirect the incident light so that it focuses onto the digital sensor. Abnormalities in a lens can cause light rays to fall short of their true location on the sensor. Likewise, the bending of the light by the lens itself can cause distortion in terms of the way in which the light is focused on the sensor. For example, a fisheye lens, which creates a wider field of view, dramatically bends the light at the peripheries of the lens to focus it on the CCD sensor. Hence, the properties of a camera have a direct effect on the resulting projection of real and virtual scenes. These effects are based on the intrinsic properties of a camera which are unique to each camera. For our purposes we refer to these as the focal length (f), skewness (s), and principle point (p_x, p_y). These intrinsic properties cause distortions in the resulting image. If these parameters are unknown, then it becomes more difficult to perform a reconstruction.

For example, if we have a point as a vector **X** in 3D space at the Cartesian coordinate (X, Y, Z) then the projection of the point on a 2D image plane is the vector **x** entailing the components (x, y). where the third dimension Z is lost from the projection. Under a pinhole camera model (roughly speaking a camera with a pinhole for a lens) as shown in Figure 3, the point **x** is where the line between the point **X** and the center of the camera intersect the image plane. When there is a lens on the camera that refocuses the light onto the sensor the path the light travels is not linear and is subject to the noise caused by the calibration matrix *K* which is formed using the intrinsic parameters of the camera and whose details are below.

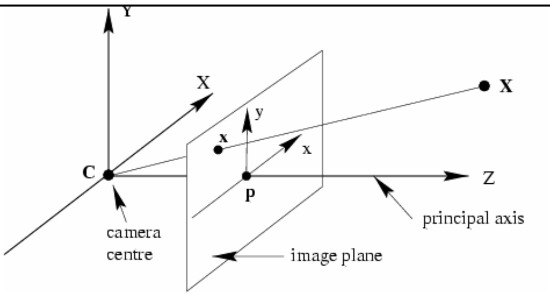


Figure 3. Camera Anatomy Showing Principle Point.

The figure depicts the relationship between the camera centre and the image plane. The point where the principle axis (optical axis) of the camera intersects the image plane is called the principle point (**p**). From:[1]

The principle point is where a line drawn orthogonal from the center of the camera (lens) intersects with the image plane and is expressed in terms of image coordinates (see Figure 3). The focal length, assuming a pinhole camera (roughly speaking, a camera with pinhole for a lens), is the distance of the orthogonal line connecting the center of the CCD sensor and the principle point. However, this is oversimplified since cameras have lenses, so the focal length is a bit more abstract. Essentially, though, it is the distance to where the light converges, or rather the distance to the focal point. Again, these intrinsic parameters can be captured in a single matrix *K,* the calibration matrix, as described by [1] and described in detail below.

The focal length and principle point cannot be expressed directly in the matrix *K* due to the relationship between the pixel's coordinates in the image and the units in which the world is measured. If a vertical line has *n* pixels and is equivalent to a horizontal line of *m* pixels such that $m=n$, then the pixels are considered *square* and the aspect ratio is one-to-one. However, if $m \neq n$ then the pixels are considered *non-square*. To capture this information in the matrix *K*, the variables m_x and m_y are introduced. They are the number of pixels in the *x* and *y* direction per world unit (e.g., m_x=100 pixels per inch in the *x* direction and m_y=98 pixels per inch in the *y* direction) and thus the matrix variables in *K* are defined as $\alpha_x = fm_x$, $a_y = fm_y$, $x_0 = m_x p_x$ and $y_0 = m_y p_y$. [1]

$$K = \begin{bmatrix} \alpha_x & s & x_0 \\ & \alpha_y & y_0 \\ & & 1 \end{bmatrix}$$

The last, unexplained variable in K is *s* which represents the skew in the image which Hartley and Zisserman state is normally zero unless the x- and y-axes are not perpendicular. [1]

The next step in camera calibration is working with the projective camera matrix (*P*) which is a 3x4 matrix that is represents both the intrinsic and extrinsic parameters of a camera. It is used to project three-dimensional homogenous points $\left(\overline{X} = \left(X, Y, Z, W\right)\right)$ onto a two-dimensional plane resulting in two-dimensional homogenous points $\left(\vec{x} = \left(x/w, y/w, w/w\right)\right)$. The homogenous component (*W)* is the integer one and used to increase the dimension of the vector for multiplication purposes. That is in order to multiply a 3D point **X** by *P* we need to upgrade the dimension of the **X** vector to 4 by adding a one. For all practical purposes we multiply each of the entries in **X** by W, but since it is one, this is not necessary to show. However, after projective **X** via

$$\vec{x} = P\overline{X}$$

we find that the w component of the resulting vector is not necessarily one, so to find the resulting 2D Cartesian coordinate we divide each of the entries by w and the quotient of the first two entries in the vector are the (x, y) coordinates.

The projective matrix is expressed as $P = K\left[R \mid \vec{t}\right]$ where *K* contains the intrinsic parameters of the camera and $\left[R \mid \vec{t}\right]$ represents the extrinsic parameters of the camera. By definition, the extrinsic parameters of the camera relate the camera pose to the world reference frame. Here, *R* is a 3x3 rotation matrix describing the rotation of the camera about its x, y and z axes, and $\vec{t}$ is a 3x1 vector representing its translation in the x, y and z directions. Hence, R and $\vec{t}$ together describe how the camera's pose is rotated first, and then translated from the origin of the world reference frame to its pose in space.[1]

If the intrinsic and extrinsic parameters are known then the camera is said to be calibrated. If only the intrinsic parameters are known, then the projective camera can be expressed as $K^{-1}P = \left[R \,|\, \vec{t} \right]$. Here, K transforms the projective matrix into the pinhole model by removing lens distortion, and the resulting projection of the three-dimensional homogenous points only depends upon the extrinsic parameters of the camera. For scene reconstruction using calibrated cameras the extrinsic parameters, R and $\vec{t}$, and hence camera matrix $K^{-1}P$, can be recovered from a sufficient number of corresponding points between two images. So, a calibrated camera plus correspondences between images allows for the estimation of a P that defines the projection of 3D world points onto the 2D image plane by only finding only R and $\vec{t}$.[1]

It is important to note that a projection can only be recovered up to scale. By analogy, this is when it cannot be determined if an image of a house is that of a doll house or a full size home. This occurs because the depth of a 3D point $\vec{X}$ from the camera (its original Z value) is removed by the projection onto the 2D image plane. The only information regarding depth that remains between images is that of parallax which is explained fully in both Chapters II and III. [1]

In summary, intrinsic camera parameters relate pixel coordinates in the image plane to the corresponding points in the camera reference frame, both planes being in $\mathbb{R}^2$. Similarly, the extrinsic parameters of a camera relate the world reference frame to the camera reference frame, which are both in $\mathbb{R}^3$. Together, the intrinsic and extrinsic properties of a camera relate the world coordinates in $\mathbb{R}^3$ to the pixel coordinates on the image plane in $\mathbb{R}^2$. This relationship is what allows for three-dimensional reconstruction.[5]

D. FEATURE DETECTION AND TRACKING

Corresponding points between images are the cornerstone of reconstruction algorithms. Correspondence comes from the idea that the points in n ($n > 1$) different images represent the projection of the same point in $\mathbb{R}^3$ onto different image planes. There are two ways to determine correspondences: by human annotation or automatically by computer. The advantage of a human annotating the images is that human capabilities still far surpass automated algorithms, so errors and outliers are minimized. However, human annotation is infeasible for real-time and near real-time applications. In order to autonomously determine correspondences between images, features in the image must be tracked between images. Having a computer do the work for us is desired but prone to errors in both estimation of the position of features as well as mismatching features.

The term "feature" is abstract in that it refers to any artifact in an image and its description. There are a vast number of ways of finding these artifacts and describing them but these artifacts are merely projections objects in 3D space onto a 2D image plane. Hence a feature could simply be a corner in an image defined by its gradient in the x and y directions. On the other hand, it could be a range of colors that define an actual object in space such as a ball painted a unique color (e.g., florescent pink). For humans, a feature can be selected from two images relatively accurately, such as pinpointing the corner of a sign in two different images. For a computer there needs to be an algorithmic method for doing this since it does not know what a sign is in the first place.

Autonomously establishing corresponding points between any two images begins by finding good features in the first image; however, there is currently no panacea for what these features are although there are some general features that are generally used. Usually the tracking algorithm that is selected is due to its performance in a particular environment and capabilities of the system using it. Ideally an algorithm is able to discern objects in an image without any prior information about them, and then proceeds to track them between frames by recognizing them regardless of transformations in their pose and illumination. Because this problem is not yet solved, the best we can do is to either know certain objects a priori that we can detect in the environment, or establish parameters that define good features for tracking purposes. In this thesis we focus on the latter; the former is beyond the scope of this thesis. The features are established in our first experiment through the use of OpenCV's GoodFeaturesToTrack method, which is based on [6]. Features have a multiplier for the max-min eigenvalue of 0.01 and minimum Euclidean distance between features of 10 pixels.

For autonomously establishing corresponding points between any two images, there are two general approaches. The difference between the two approaches occurs in how the second image is processed. One method is to repeat the same processing on the second image as was performed on the first image, and then establish correlations between the set of features from the first and second images and consider these to be corresponding points. The alternative method, instead of searching the second image independently for features, is to track the initial features into the subsequent frame.

There are many problems that haunt feature tracking algorithms some of which are listed below:

a. Features can move out of the field of view.
b. The illuminance affecting a feature can change.
c. An image can have repetitive patterns.
d. Features can be blurred by camera motion.
e. Tracked points may be occluded in one of the images.

In this thesis we use OpenCV's GoodFeaturesToTrack algorithm. It ultimately extracts features from an image that can be tracked from frame to frame. In order to accomplish this it uses the Sobel operator, which uses both vertical and horizontal kernels to calculate the derivative of the image's signal at each pixel in the image, given by

$$S_x = \begin{bmatrix} -0.5 & 0 & +0.5 \\ -1 & 0 & +1 \\ -0.5 & 0 & +0.5 \end{bmatrix} \quad S_y = \begin{bmatrix} +0.5 & +1 & +0.5 \\ 0 & 0 & 0 \\ -0.5 & -1 & -0.5 \end{bmatrix}.$$

[7]

The response of a neighborhood to the kernel determines its derivative in the x and y direction respectively. The actual gradient, theta, of the illumination of the pixel is calculated by

$$\theta = \arctan\left(\frac{S_x}{S_y}\right).$$

[7]

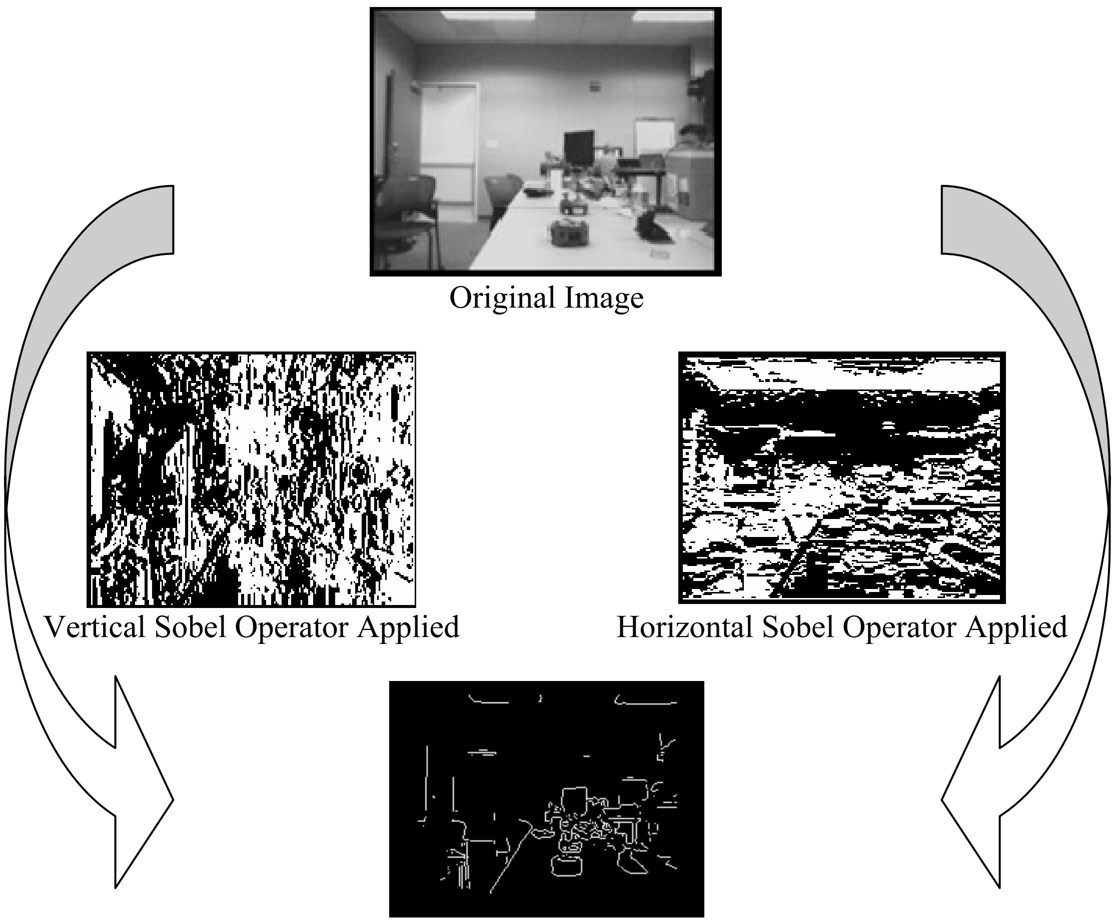

Original Image
Vertical Sobel Operator Applied
Horizontal Sobel Operator Applied
Gradient Calculated, Threshold Applied

Figure 4. Sobel Operator in Action.

The above diagram shows how an image is transformed using the Sobel operator to find edges in an image.

Once the gradient of each pixel is determined, a threshold can be applied and the edges extracted as shown in Figure 4. However, once the derivative (D_x, D_y) at each pixel is calculated, the feature detection algorithm computes the eigenvalues and eigenvectors of the covariance matrix $C = \begin{bmatrix} \sum D_x^2 & \sum D_x D_y \\ \sum D_x D_y & \sum D_y^2 \end{bmatrix}$ where the summation is over the block of pixels being examined by the kernel. The larger the eigenvalues of *C,* the better the feature. One caveat is that a feature is rejected if it is too near another. [7]

Having found features in one image, the next step is to find a corresponding feature in a subsequent image. The algorithm we use tracks features to the second image by determining its motion between images (optical flow). The OpenCV method used for our experiments is based on the Lucas Kanade feature tracking, using image pyramids. The complete details are beyond the scope of this thesis but we will provide a general overview. [8].

Based on the algorithms used, an image pyramid is a progressively smaller set of images where the value of the pixel in the next higher level of the pyramid is the average of the 2x2-pixel array below it. The advantage of an image pyramid for feature tracking is that it provides global information where otherwise the operator is constrained to searching for a corresponding point locally. By analogy, imagine it is the difference between searching for a missing person on foot and by helicopter. The helicopter gives a global perspective of where people could be, where the person on foot can only search locally. Together, however, if one member of the search party is on foot and another in a helicopter, the helicopter borne person can find candidates who could be the missing person quickly, and the searcher on the ground can determine the identity of the candidate. In feature tracking each feature has a signature that can be used to identify it, and candidates are searched for by the pyramid method such that the optical flow of each level of the pyramid gives clues as to the location of the corresponding feature.[8]

E. FUNDAMENTAL MATRICES AND EPIPOLAR GEOMETRY

At the core of a 3D reconstruction is the relationship between the features that are tracked between two images. To find the relationship a fundamental matrix must be estimated which relates the points to each other by relating a point in one image to a line in the other, which in turn the corresponding point should lie on this line. This is what is referred to as epipolar geometry, which specifically refers to the relationship between corresponding points in two different projections of the same three-dimensional scene. It only depends upon the camera's internal parameters (intrinsic parameters) and relative pose (external parameters) and is captured entirely in a 3x3 matrix called the fundamental matrix (*F*). The inputs to the various algorithms that estimate a fundamental matrix are corresponding points (***x***⇔***x'***) between two images where ***x*** and ***x'*** are vectors representing the Cartesian coordinate (*x, y*) of the same feature in two different images. The coordinates are usually normalized for each image independently so that they are all a certain distance from the centroid of the points. Simplified, this normalization avoids computational artifacts due to the arbitrary pixel location units.

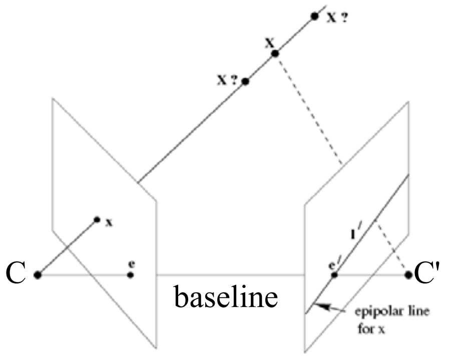


Figure 5. Epipolar Geometry.

Diagram of the epipolar geometry, where point *x* has a corresponding point on the epipolar line *l'*, which must pass through the epipole *e'*. The different triangulated points, *X*, are the possible results based upon where the corresponding point is located on the epipolar line. From: [1]

The epipolar geometry between two images is illustrated in Figure 5. Given two camera centers (C, C'), the line connecting them is called the baseline. Epipolar lines in related images are defined by the fundamental matrix such that $Fx=l'$ and $F^Tx'=l$. They are called "epipolar" since all the lines defined in this manner intersect the image plane's epipoles (e,e'), or rather radiate out from this point. The epipole is where the baseline intersects the image plane and is considered to be the projection of the other camera's center. The Cartesian coordinate of the epipole e is the null space for F in the first image since all lines defined by F for the image run through e. Likewise, e' is the null space for F^T. Hence, the epipoles are defined algebraically as $F\boldsymbol{e}=0$ and $F^T\boldsymbol{e}'=0$ for the respective images. To find the non-trivial solution to the null vectors e and e', we can use Singular Value Decomposition (SVD) such that [u d v] = SVD(F), where u is the left orthogonal matrix and v is the right orthogonal matrix. The last column of u and v are the epipoles e and e' respectively.

The relationship between corresponding points, as defined by the epipolar geometry, is that point x corresponding point x' must lie on the epipolar line defined by Fx. Hence, the fundamental matrix relates points in one image to epipolar lines in the second image where a corresponding point can be found. The role of the fundamental matrix as it relates to accurately predicting the epipolar geometry for finding corresponding points is one object of scrutiny in this thesis.

A more specialized version of the fundamental matrix is the essential matrix which relates two views in much the same way as the fundamental matrix, but the relation between the images is solely dependent on the extrinsic parameters of the camera due to the fact that the calibration matrix K is known. The relation between the essential matrix and the fundamental matrix is $E = K'^T FK$. If the calibration matrices are the same for both images, then $K'=K$. [1]

F. MEASURING ERROR IN AN EPIPOLAR GEOMETRY

We need a method for determining how well the calculated fundamental matrix fits the epipolar geometry between two scenes. We know that a point in one images is mapped to a line in the other by way of the fundamental matrix such that $Fx=l'$ and

$F^T x' = l$. To determine if the corresponding point actually falls on the line we can use linear algebra and multiply *lx* and $x'^T l'$. If the product is zero then the point lies on the line. Through substitution we find that both are equivalent and can be simplified into a single equation $x'^T F x = 0$. However it is not intuitive as to what it means if the product is nonzero. We consider these nonzero numbers to be errors and this section details what this error value means.

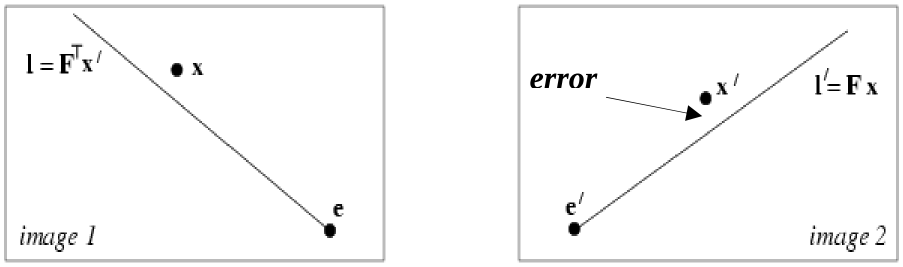


Figure 6. Error in the Epipolar Geometry.

Illustration showing the error in the epipolar geometry between two images defined as "$x'^T F x = error$." From: [1]

In Figure 6 above, the error in the epipolar geometry is illustrated showing how the *x* and *x'* points don't fall on their respective epipolar lines. It is naïve to think that the result of the equation $x'^T F x$ is the number of pixels between the point and the line. If the solution is indeed nonzero, then the units of such a measurement are abstract and it is best explored by example.

The equation of a line is $ax+by+c=0$. So if we set {a=2, b=-10, c=1200} then the vector representing that line is $l=[2\ -10\ 1200]^T$. So the measure of error for any one point in a 400x400 image ($(x,y) \mid x \in \{1:400\}, y \in \{1:400\}$) between line *l* can be expressed as follows:

$$\begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} 2 \\ -10 \\ 1200 \end{bmatrix} = error$$

If we do this for every point in the entire image, and then normalize the errors between 0 and 1 we find Figure 7. In the figure, the bright line with a steep negative slope is the sample epipolar line. The lighter the pixels is, the greater the error. Hence, we can see that there is a smooth gradient of error as we move from the epipolar line. But this still does not explain what units this error is measured in.

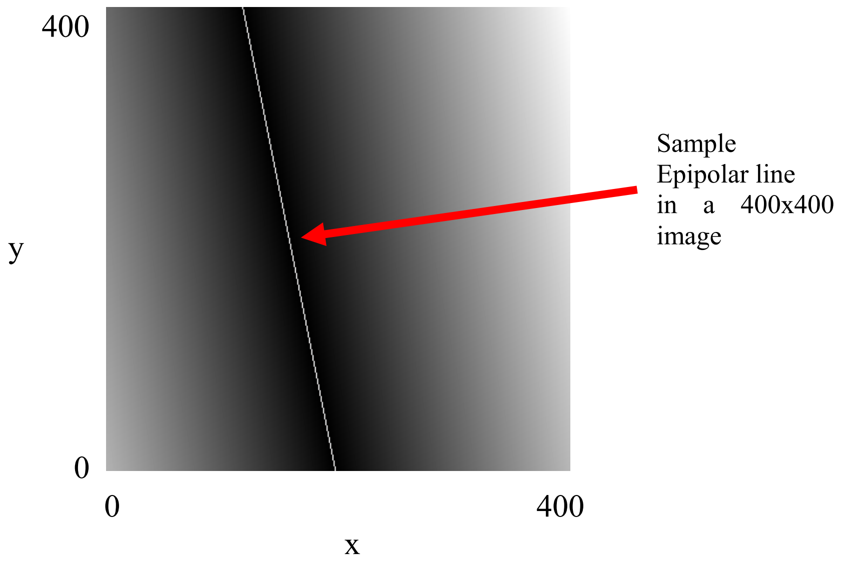


Figure 7. Graphical Depiction of Epipolar Error.

Image showing the error of every pixel as it relates to the epipolar line defined as 2a-10b+1200=0 where the pixel values range from 0 to 1. (one being the brightest with the most error). The distribution of the errors is dependent upon the coefficients of a and b.

To determine the units, we look at individual points. The point (26, 30), lies on the line, and the homogenous representation of the point multiplied by the equation of the line finds the error to be zero as expected. However, the point (26, 29) has an error of -2 and the point (27, 30) an error of -10. This is the part that is not intuitive. Here the points are one pixel away from the line yet the errors are large. In fact, the errors are a function of the line's *a* and *b* coefficients where $ax+by+c=0$. In order to convert the error values to actual pixel distances in both the x and y directions, we can divide by the coefficients *a* and *b* of the original line. An example is shown below.

$$\vec{l}\vec{x} = error$$

$$\begin{bmatrix} 2 & -10 & 200 \end{bmatrix} \begin{bmatrix} 27 \\ 30 \\ 1 \end{bmatrix} = -10$$

$$\Delta x = \frac{error}{b}, \Delta y = \frac{error}{a}$$

$$\Delta x = \frac{-10}{-10} = 1$$

$$\Delta y = \frac{-10}{2} = -5$$

Here a positive 1 for delta-x is one pixel in the positive x direction. A -5 for delta-y is 5 pixels in the negative y direction from the line. Hence, the value of the error found by "$x'^T Fx = error$" contains components in the x and y directions.

For our purposes, we are disinterested in the actual pixel distances between the point and the line. If different fundamental matrices are supposed to estimate the same epipolar geometry derived from the same corresponding points, then the estimates are directly comparable by looking at the errors each generates.

G. ROBUST REGRESSION

Regression is the standard method to fit a model to an observed dataset, and tracked features define our dataset that is the input to our vision algorithms. That being said, these features are not always tracked correctly and it is quite possible that some of the features were wrongly identified in the subsequent frame. These poorly tracked features plague our ability to perform a proper regression on the dataset. Under a normal regression, such as linear least squares, even a single large outlier included can and will have dramatic effects on the resulting epipolar geometry as seen in Figure 8. The standard way of dealing with outliers is to use a robust regression which attempts to identify corresponding features that are mismatched between images and classifies them as outliers. [9]

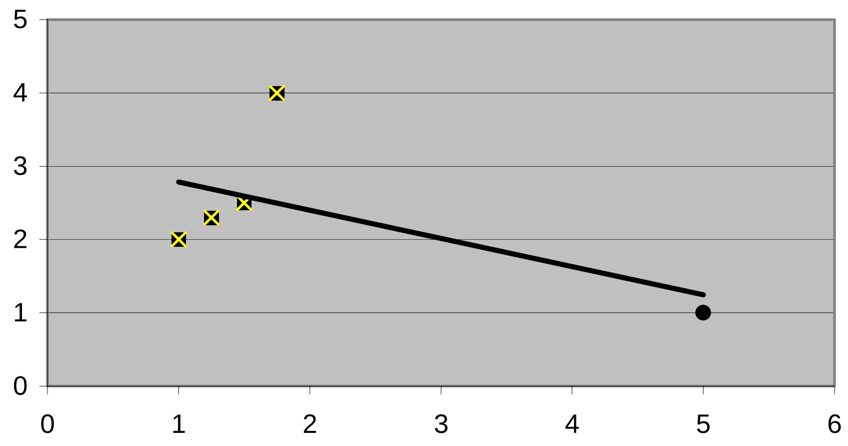


Figure 8. Outlier Effect.

Depiction of how a single outlier (the circle) can have dramatic effect on the least-squares regression.

"Robust" refers to the ability of the regression to recover from unexpected conditions, i.e., outliers. According to [9], the modern versions of robust regression stems from minimax optimization which is related to game theory and is used in IBM's Deep Blue. In computer vision, RAndom SAmple and Consensus (RANSAC) is the most popular. In general, it randomly chooses a subset of the data points, performs regression to determine a candidate model, also referred to as the *influence function*. The influence function is then scored by the number of points that fall within a threshold distance, also known as a breakdown point. If the candidate model does not create enough inliers, a new subset of points is randomly chosen and the process repeated. Once the number of inliers exceeds the breakdown point, or a set number of iterations has been tried, the iterative process will exit and return the last or best influence function. [9]

An alternative method of measuring a model that does not require any a priori knowledge to set thresholds, is to use the Least MEDian of Squares (LMEDS). As with RANSAC it chooses subsets of the data to create a candidate models, but instead of classifying the data by a threshold distance to that model, it determines the median distance of all points to that model. The model with the lowest (least) median is returned. LMEDS fails when more than 50% of the data are outliers. [1]

The number of algorithms that have been proposed to deal with outliers in a robust manner is rather large. The list includes Least Absolute Values (L1 regression), M-estimators, GM-estimators, Least Trimmed Squares, Weighted Least Squares, etc. For more information on the details of these and others please see [9].

The advantage of using a robust estimator should be obvious; however, as with anything there are downsides to robust estimators as well. They

- are not guaranteed to find the best solution;
- are not guaranteed to find a quality model if one exists;
- can be computationally expensive; and
- can require a priori knowledge of the data.

H. NON-LINEAR LEAST SQUARES AND LEVENBERG-MARQUARDT

Having identified point correspondences as inliers and outliers by using a robust method, the next problem faced is that the noise from feature tracking and estimations of the fundamental matrix leave us with a very complex system to be modeled whose error cannot be measured linearly as in ordinary least squares. In other words, the cost function for determining the error between a data point and the model in our system is not linear. To deal with such cases we require the use of nonlinear regression. This is often confused with the concept of fitting data to a nonlinear model, but even a conic can have data fitted to it by linear least squares regression. For our purposes the cost function is based on the Taylor expansion and is an approximation to the first order geometric error: *Sampson Distance*. [10]

Sampson distance is derived from research done by Bookstein who presented seminal work on conic fitting which proposed that data could be fit to a conic by a rank-deficient generalized eigenvalue system $D^T D\vec{a} = \lambda C\vec{a}$. Where D is a matrix of unknowns, and C is a matrix that places constraints on the system and the vector ***a*** is our datapoint. In the case of Bookstein, the constraint was that the norm of vector ***a*** must equal 1. [11]. Sampson improved this by proposing an iterative method that better approximates the first order geometric error, where Bookstein's was based upon an algebraic distance measure. Sampson distance itself is an approximated maximum likelihood cost function. [10]

Although Sampson describes the non-linear cost function for an iterative process, there are various iterative approaches that can be used. The most prevalent non-linear algorithm used in reconstruction algorithms is Levenberg-Marquardt (LM). It is non-linear from the view point that the cost function is non-linear with respect to the unknown parameters and has the form $y = f\left(\vec{x} : \vec{\beta}\right) + \varepsilon$, such that the vector β holds the unknown parameters. However, the method of least squares is still the basis to determine the unknown parameters by minimizing the errors determined by the cost function, hence the LM algorithm is categorized as a non-linear least squares algorithm. The reason for using the LM algorithm is that data can be fit to a broad range of functions and is not limited to linear cost functions only; however, the cost of the freedom is that the process is iterative which increases the computation time. [10]

The LM algorithm is based upon both the gradient descent method and Gauss-Newton minimization. There are various advantages to using the LM algorithm over other iterative approaches, but the most notable is the fact that it can converge on a local minimum quickly similar to the Gauss-Newton function due to LM’s relation to gradient descent, even when the Gauss-Newton function fails. The reason is that the Gauss-Newton function requires that an initial estimate of the unknown parameters be fairly close to the local minimum. The further from the local minimum, the more likely the Gauss-Newton function will fail to move towards it due to its reliance on the estimation of the Hessian matrix. The gradient descent method, on the other hand, will always move in the direction of the greatest local gradient. If we are relatively far from the local minimum where the gradient is only slight, the gradient descent method will iteratively advance the estimated solution towards the local minimum. The LM algorithm will do the same due to its relationship to gradient descent, except as the estimate gets closet to a local minimum, the Gauss-Newton minimization component of the LM algorithm will "take over" advancing the estimation quickly to the minimum than what the gradient descent would. [1]

1. Optimization Using Sampson Distance

The first point at which an iterative process is useful is after the initial estimation of the fundamental matrix. Being estimates, the true values of the fundamental matrix are the unknowns in our least squares problem. To determine the current quality of the fundamental matrix we need a cost function. The recommended cost function is Sampson Distance.[1] It was first used as a means for fitting points to a conic in replacement to a iterative process by measuring the orthogonal distance to the conic. [10] In regards to three-dimensional reconstruction algorithms, it is advantageous to use this measurement as a cost function. The reason is that it estimates a reasonable approximation to the first-order geometric error between a measured point (*x,y*) and the current estimate of our model of the epipolar line. Without Sampson distance, the geometric error could be found for each iteration but it would require another iterative process adding another level of complexity to the overall algorithm. [12]

The equation for determining the Sampson distance, as it relates to the cost function for the LM algorithm, is

$$\sum_i \frac{\left(x_i^{'T} F x_i\right)^2}{\left(Fx_i\right)_1^2 + \left(Fx_i\right)_2^2 + \left(F^T x_i^{'}\right)_1^2 + \left(F^T x_i^{'}\right)_2^2}$$

[1]

where the Sampson distance is summed over all corresponding pairs ($x_i \Leftrightarrow x_i'$) using the current estimate of the fundamental matrix (*F*). The numerator is the square of the familiar relationship between the corresponding points and the epipolar geometry. The denominator is a reasonable approximation of the Hessian of the second derivative of the Taylor series with respect to the unknown coefficients and is represented as the product of $J^T J$ such that J is the Jacobian matrix. However this form of the Sampson distance is different from that presented by Ma et al,

$$d^j = \frac{\left(x_2^{jT} F x_i^j\right)^2}{\left\|\widehat{e_3} F x_1^j\right\|^2 + \left\|x_2^{jT} F \widehat{e_3}\right\|^2}$$

[12]

where e$_3$ is the cross product of e$_1$ and e$_2$.

The epipoles are estimated such that e = $(\alpha, \beta, -1)^T$ and e'=$(\alpha', \beta', -1)^T$. With these estimations we then parameterize the fundamental matrix in such a way that it remains rank two. There are three proposed parameterization of the fundamental matrix offered by Hartley and Zisserman. We chose the method that uses both epipoles and the fundamental matrix to get the following form where {*a ,b, c, d*} are the initial upper left entries in F: [1]

$$F = \begin{bmatrix} a & b & \alpha a + \beta b \\ c & d & \alpha c + \beta d \\ \alpha' a + \beta' c & \alpha' b + \beta' d & \alpha' \alpha a + \alpha' \beta b + \beta' \alpha c + \beta' \beta d \end{bmatrix}$$

The resulting F is then vectorized in row-major order to create a 9x1 column vector for input into the LM algorithm.

LSQNONLIN is a nonlinear solution solver in MATLAB which can be called with the Levenberg-Marquardt option turned on. The parameterized F is passed into the method as the initial point along with the cost function used to determine the error in the current estimate. One thing to note is that the LSQNONLIN implementation seeks to minimize the sum of the squared errors in the returned matrix, so the cost function merely needs to return the output of the function below for each corresponding pair as a single vector.

$$\frac{\left(x_i^{'T} F x_i\right)^2}{\left(Fx_i\right)_1^2 + \left(Fx_i\right)_2^2 + \left(F^T x_i'\right)_1^2 + \left(F^T x_i'\right)_2^2} = error_i$$

Upon completion of the iterative process, the algorithm returns the refined 9x1 vector which is reshaped into the 3x3 fundamental matrix.

2. Optimization Using Gold Standard Method

The second point in the flow of a reconstruction where and iterative process is advised is after a projective reconstruction is completed, i.e., when the 3D projective points have been triangulated. In general, a projective reconstruction algorithm is as follows:

- Estimate a fundamental matrix from corresponding points between images.
- Extract epipoles from the fundamental matrix such that $Fe = 0$ and $F^T e' = 0$.
- Assume the projection matrix for the first camera (*P*) is a zero augmented identity matrix $\left(P = \left[I \mid \vec{0}\right]\right)$, and calculate the projection matrix for the second image (*P′*) such that $P' = \left[\left[e'\right]_x F \mid e'\right]$. [1]
- Using the estimated camera matrices (*P* and *P′*) and the original corresponding points $\left(\vec{x}_i \Leftrightarrow \vec{x}_i'\right)$ triangulate to determine three-dimensional projective points X_i. However, these points are neither Euclidean nor metric. They are projective and as such any relationship we know to ground truth point in Euclidean space cannot be used to measure error.

Once X_i is triangulated for all *i*, the camera matrices *P* and *P′* can project the points back onto the two-dimensional image plane creating the reprojected corresponding points $\left(\hat{x} \Leftrightarrow \hat{x}'\right)$. H&Z advise using the Gold Standard method for minimizing the geometric distance in the reprojection of the three-dimensional points by *P* and *P′*. The cost function for each iterative step is the Euclidean distance between corresponding points where the distance is given by

$$\vec{x} = \left[x_1, y_1, 1\right]^T \qquad \hat{x} = \left[x_2, y_2, 1\right]^T$$

$$dist = \sqrt{\left(\left(x_1 - x_2\right)^2 + \left(y_1 - y_2\right)^2\right)}$$

[1]

Essentially this is an indirect optimization of the fundamental matrix which is supposed to further reduce the errors in *F's* estimate of the epipolar geometry between the two scenes. The iterative process adjusts both the 3D projective points and the estimated camera matrix *P'* until the local minimum is approximated. The function then returns the new camera matrix P'_{new}. This is used to change our estimate of the fundamental matrix such that $P'_{new}=[M|\mathbf{t}]$ and $F_{new}=[\mathbf{t}]_x M$, where $[\mathbf{t}]_x$ is the skew-symmetric form of the last column of P'_{new}, and M is the first 3x3 matrix of P'_{new}.

For the problem at hand, the minimization of the square root of the sum of squares is equivalent to the minimization of the sum of squares so the distance vector that is returned by the cost function is $\left((x_1-x_2)^2+(y_1-y_2)^2\right)$ for each point *i*. Even though the goal is to minimize the error to our original point correspondences, their Cartesian coordinates are noisy. An alternative approach to using the initial point correspondences as the target of optimization, is to use the ideal points that are predicted by the epipolar geometry such that they lie on the epipolar lines for the current estimated *F* for each iteration.[12]

III. LITERATURE REVIEW

In literature there are efficient ways to produce the fundamental matrix when the type of camera motion is known (e.g., pure planar). Hartley and Zisserman[1] express a few ways to find the fundamental matrix under different motions and extend their comments to discuss what insight this gives to calculation of the fundamental matrix under general motion. However, it has not been looked at specifically how different camera motions effect the general case (unconstrained camera motion), and how well the different algorithms and implementation perform under different camera motions. The seed for bringing about this research is from Ma's comments on Parallax.[12]

A. START OF IT ALL

When machines first began to be called computers, researchers like Norbert Wiener at MIT wondered what the capabilities of these machines would be. How would they complement and supplement human capabilities? Of interest was if these new machines could one day have the same visual capabilities. [13]

The cerebral cortex is the outermost part of our brain where there are about 20 billion neurons that control most complex human functions. One of these functions is our visual capabilities. It is estimated that over half of the cerebral cortex is dedicated to vision. [13] Yet, in the sixties the computer scientist community was under the impression that computer vision could be solved simply based on the paradigm that humans do it so easily. [1] Now, almost fifty years later, we are still trying to solve many of the tasks especially those to perform vision in a general manner with no domain information.

B. PROJECTIVE RECONSTRUCTION ALGORITHM

After the many years of research, researchers have been able to accomplish what is termed a projective reconstruction. This is when features tracked between scenes have been triangulated to recreate the 3D scene that was projected into the images. The recovered 3D scene is in projective space and appears to be the original scene when viewed from the right perspective and can be done using images from uncalibrated cameras. The general overview of the algorithm is in Figure 9. A more detailed and technical version is found in Figure 10.

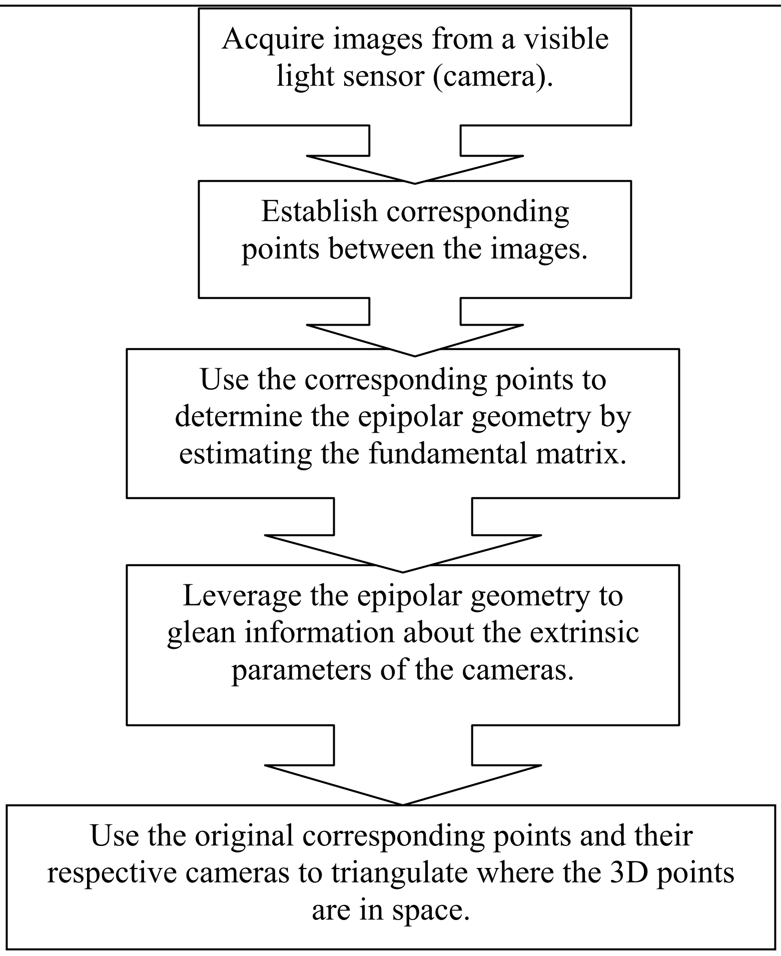
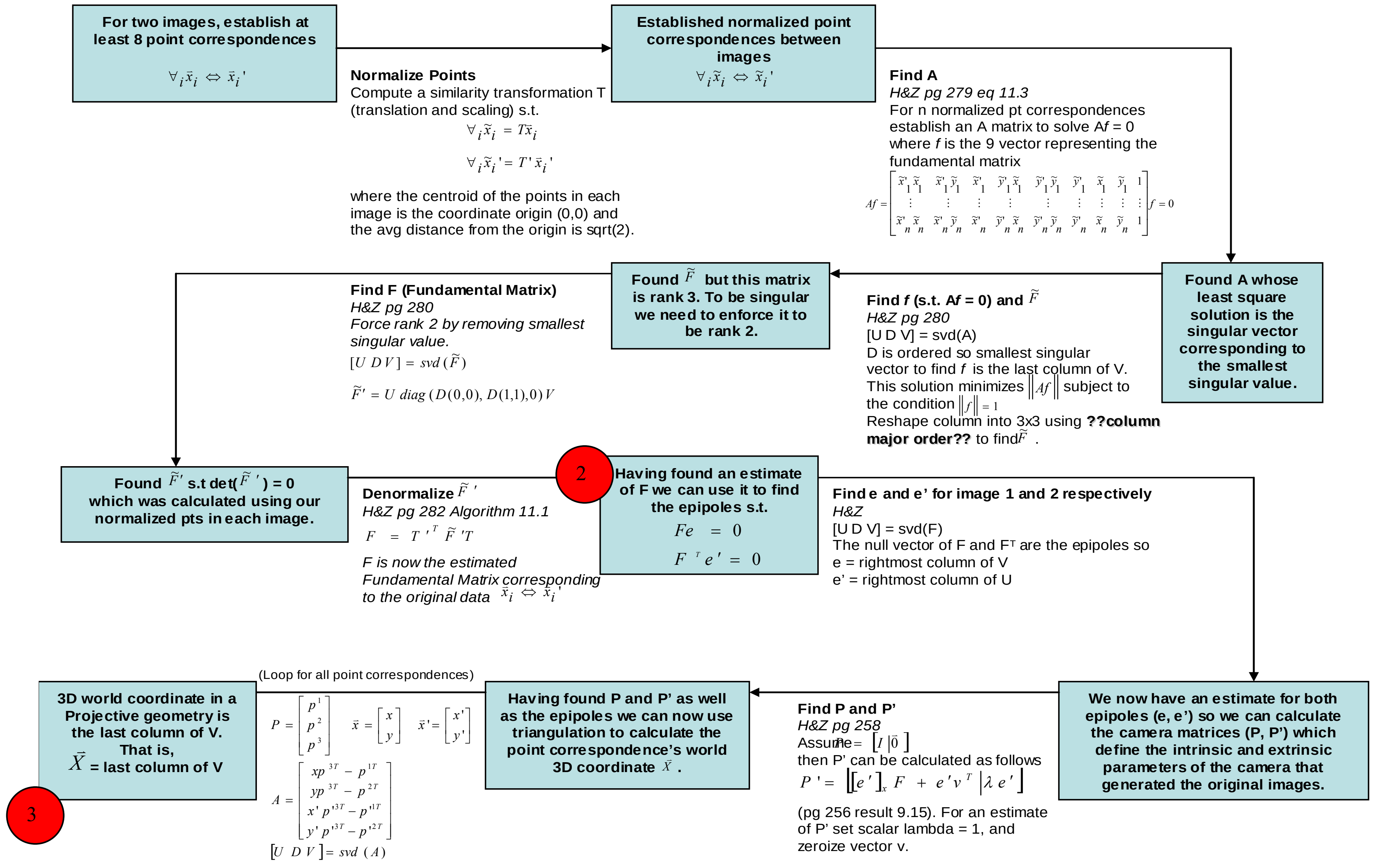

Acquire images from a visible light sensor (camera).
Establish corresponding points between the images.
Use the corresponding points to determine the epipolar geometry by estimating the fundamental matrix.
Leverage the epipolar geometry to glean information about the extrinsic parameters of the cameras.
Use the original corresponding points and their respective cameras to triangulate where the 3D points are in space.

Figure 9. General Flow Of A Projective Reconstruction.

The input to the algorithm are the corresponding points between two images, and the output is a projective reconstruction of the rigid scene. We present this algorithm without comment now so it can be referred to during the reading. Chapter II covered many of the background topics, and several more will be covered later in this chapter.

Note that the red circles in Figure 10 represent at which stage in the algorithm experiment 2 and 3 take place.



For two images, establish at least 8 point correspondences

$\forall_i \bar{x}_i \Leftrightarrow \bar{x}_i'$

Normalize Points
Compute a similarity transformation T (translation and scaling) s.t.

$$\forall_i \tilde{x}_i = T\bar{x}_i$$

$$\forall_i \tilde{x}_i' = T'\bar{x}_i'$$

where the centroid of the points in each image is the coordinate origin (0,0) and the avg distance from the origin is sqrt(2).

Established normalized point correspondences between images

$\forall_i \tilde{x}_i \Leftrightarrow \tilde{x}_i'$

Find A
H&Z pg 279 eq 11.3
For n normalized pt correspondences establish an A matrix to solve A*f* = 0 where *f* is the 9 vector representing the fundamental matrix

$$Af = \begin{bmatrix} \tilde{x}'_1\tilde{x}_1 & \tilde{x}'_1\tilde{y}_1 & \tilde{x}'_1 & \tilde{y}'_1\tilde{x}_1 & \tilde{y}'_1\tilde{y}_1 & \tilde{y}'_1 & \tilde{x}_1 & \tilde{y}_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \tilde{x}'_n\tilde{x}_n & \tilde{x}'_n\tilde{y}_n & \tilde{x}'_n & \tilde{y}'_n\tilde{x}_n & \tilde{y}'_n\tilde{y}_n & \tilde{y}'_n & \tilde{x}_n & \tilde{y}_n & 1 \end{bmatrix} f = 0$$

Found A whose least square solution is the singular vector corresponding to the smallest singular value.

Find *f* (s.t. A*f* = 0) and $\tilde{F}$
H&Z pg 280
[U D V] = svd(A)
D is ordered so smallest singular vector to find *f* is the last column of V.
This solution minimizes $\|Af\|$ subject to the condition $\|f\| = 1$
Reshape column into 3x3 using **??column major order??** to find $\tilde{F}$.

Found $\tilde{F}$ but this matrix is rank 3. To be singular we need to enforce it to be rank 2.

Find F (Fundamental Matrix)
H&Z pg 280
Force rank 2 by removing smallest singular value.

$$[U\ D\ V] = svd(\tilde{F})$$

$$\tilde{F}' = U\ diag(D(0,0),\ D(1,1), 0)\ V$$

Found $\tilde{F}'$ s.t det($\tilde{F}'$) = 0 which was calculated using our normalized pts in each image.

Denormalize $\tilde{F}'$
H&Z pg 282 Algorithm 11.1

$$F = T'^{T}\ \tilde{F}'T$$

F is now the estimated Fundamental Matrix corresponding to the original data $\bar{x}_i \Leftrightarrow \bar{x}_i'$

2

Having found an estimate of F we can use it to find the epipoles s.t.

$$Fe = 0$$

$$F^{T}e' = 0$$

Find e and e' for image 1 and 2 respectively
H&Z
[U D V] = svd(F)
The null vector of F and F^T are the epipoles so
e = rightmost column of V
e' = rightmost column of U

We now have an estimate for both epipoles (e, e') so we can calculate the camera matrices (P, P') which define the intrinsic and extrinsic parameters of the camera that generated the original images.

Find P and P'
H&Z pg 258
Assume P = $[I\,|\,\vec{0}\,]$
then P' can be calculated as follows

$$P' = \left[[e']_{x}F + e'v^{T}\,|\,\lambda e'\right]$$

(pg 256 result 9.15). For an estimate of P' set scalar lambda = 1, and zeroize vector v.

Having found P and P' as well as the epipoles we can now use triangulation to calculate the point correspondence's world 3D coordinate $\vec{X}$.

(Loop for all point correspondences)

$$P = \begin{bmatrix} p^{1} \\ p^{2} \\ p^{3} \end{bmatrix} \quad \vec{x} = \begin{bmatrix} x \\ y \end{bmatrix} \quad \vec{x}' = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

$$A = \begin{bmatrix} xp^{3T} - p^{1T} \\ yp^{3T} - p^{2T} \\ x'p'^{3T} - p'^{1T} \\ y'p'^{3T} - p'^{2T} \end{bmatrix}$$

$$[U\ D\ V] = svd(A)$$

3D world coordinate in a Projective geometry is the last column of V. That is,
$\vec{X}$ = last column of V

3

Figure 10. Projective Reconstruction Flow Chart.

The point in the algorithm that the second and third experiment take place is depicted by the numbered circle.

C. PARALLAX AND CAMERA MOTION

Different camera motions create different types of levels and types of parallax between images. Hence, our thesis can be interpreted as comparing how the different levels and types of parallax shift between images effects the errors in the estimation of an epipolar geometry. By definition, parallax (also known as motion parallax, trigonometric parallax, and stellar triangulation) is the angle measurement in arc seconds between two different stationary objects at different depths between two viewpoints. This can be experienced by holding your thumb out in front of you and alternatively opening and closing either eye. Your thumb experiences relative movement to the background and seemingly jumps between positions as you alternate eyes. Similarly, when points are projected onto the image plane, the points closest to the image plane will experience a greater jump, or rather parallax shift, than points that are further away.

In astronomy, parallax is used to determine distance to objects in the universe. Using the measured distance to the sun (*R*), and leveraging earth's rotation around the sun, a picture of a star can be taken six months apart when Earth is on either side of the sun (similar to our two eyes on opposite sides of our head). The distance (*D*) to the star is calculated by determining the parallax angle (θ) from the parallax shift between the subject star and other stars in the night sky. This calculation is formulated as $\tan(\theta) = R / D$,or rather $D = \tan^{-1}(R)$. [14]

The use of parallax in 3D reconstruction was explored in the 90s by way of parallax geometry. Lately there have been very few articles addressing this approach, and more attention has been on the use of epipolar geometry. There are methods for determining the fundamental matrix from parallax geometry, but we have been unable to find any study that looks specifically at the effect parallax has on estimating the fundamental matrix and, in turn, epipolar geometry. Articles that use parallax geometry, similar to epipolar geometry, include Boufama, et al., in 1995 which estimated the epipolar geometry between two views using virtual parallax and a homography that linked the two views.[13] And as recently as 2005 the Trifocal tensor (similar to the fundamental matrix) was estimated using parallax by the Institute of Computer Science, Foundation for Research and Technology - Hellas (FORTH) [13]

Our interest in parallax stems from comments in Yi Ma's book "An Invitiation to 3D Vision." Ma says explicitly that the *"translation in the image must be greater than zero"* as well as that there needs to be "'*sufficient parallax' for the algorithm to be well conditioned."*[12] However, he gives no reference as to where this information came from and we are unable to find a detailed study in literature as to the direct nature of the effect parallax has on the estimations of the epipolar geometry. What we do know is that there are claims that the less parallax between images, the more difficult it is to calculate the epipolar geometry (i.e., the less translation in camera motion). In fact, there exists a geometric relationship derived from the parallax displacements of points with respect to an arbitrary planar surface, which does not involve epipolar geometry. Furthermore, the dual of the epipole can be found from parallax motion, where the epipole relates many points between two images, its dual relates two points over multiple images. Furthermore, the dual is invariant with respect to camera motion where the epipole is invariant to scene structure. [15]

Further analysis of parallax geometry is beyond the scope of this thesis, but it is obvious in our literature review that in the 1990s there was a divergence in 3D scene analysis where one line of thought focused on using epipolar geometry and another using parallax geometry. The use of parallax has dwindled but there may still be some validity to its use, although, again, beyond the scope of this thesis.

D. VISION BASED SLAM ALGORITHMS

The following algorithms all provide approaches to vision based SLAM that can benefit from our research in that a more efficient approach to their solution can come from ensuring that their algorithms are made efficient with respect to different camera motions.

1. MonoSLAM

MonoSLAM[16] is an approach that uses a monocular camera to conduct SLAM by way of a sparse map which means the actual map itself is a point cloud of the 3D Euclidean location of the tracked features. Essentially the output is a 3D occupancy grid. Using quaternions to define the features, the algorithm is able to both store them and reacquire them when they come back into view. The focus of the algorithm is a real-time, monocular vision based, SLAM algorithm. An added feature of the MonoSLAM sparse map over a normal occupancy grid is that the features correspond to a covariance matrix which defines the probability of the true location of the feature.. The problem with Davison's approach is that it requires that the algorithms be bootstrapped by an initial calibration of something known, i.e., a black square of known size. The ideal system would not require a bootstrapping procedure and an uncalibrated camera but until the notion of relative scale is solved for uncalibrated cameras, the bootstrapping is necessary. However, we believe our results will complement MonoSLAM by providing more details about the accuracy of epipolar geometry estimates. Images from a [16]

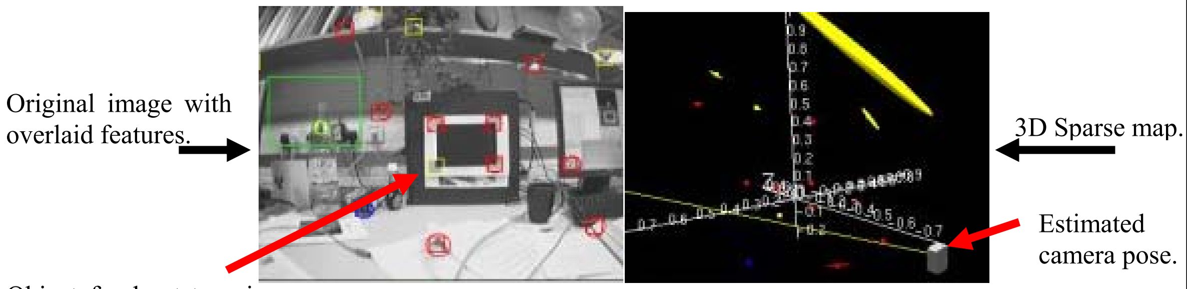

Original image with overlaid features.
Object for bootstrapping the algorithm.
3D Sparse map.
Estimated camera pose.

Figure 11. MonoSLAM.

From: [16]

2. FastSLAM Based MonoSLAM

FastSLAM is an algorithm developed for traditional range based sensor SLAM that uses particle filters. A particle represents a probabilistically weighted pose of the robot, and each is composed of a historic path estimate and a covariance matrix coupled with a set of estimators of individual feature locations. New readings update the particles and cull those that are probabilistically unlikely. [17] At the Computer Vision and Pattern Recognition (CVPR) conference in 2006, Eade and Drummond have developed a SLAM algorithm based on FastSLAM using a single camera as a sensor.[18] They have demonstrated a camera traversing a circular pattern in a three-dimensional area, mapping multiple features with a successful closing of the loop. The demonstration showed the applicability of traditional SLAM algorithms to vision based SLAM, but gives neither full details about the accuracy and precision of the location of the features tracked in three-dimension nor about the accuracy and precision of the camera pose. Their work is directly related to ours in that our focus is on increasing the accuracy of estimations in epipolar geometry which is correlated with accuracy in three-dimensional localization that increases the rate of convergence for the location of a point in a sparse map.

3. SLAM from SFM

Masahiro Tomono presented a algorithm for solving monocular vision based SLAM based on 3-D reconstruction algorithms (SFM).[19] The paper he shows how to generate an A matrix that satisfy's degeneracy conditions of the fundamental matrix so that traditional structure from motion algorithms can be applied to SLAM. Hartley and Zisserman (H&Z) discuss how the fundamental matrix fails if the points are coplanar, or if the points are from camera motion that lacks translation. [1] Starting with the third image, Tomono uses the features from the first and k^{th} images (current image) in the sequence to find the matrix A using the Kronecker tensor product as in the 8-Point Algorithm. Taking the SVD of the A matrix, the square root of the product of the seventh and eighth entries in the singular value matrix are compared against a threshold. When the value exceeds the threshold the algorithm continues on to determine the structure of the scene. The motivation behind this comes from H&Z's argument on the degeneracy of the fundamental matrix when the camera motion lacks translation. This relates directly to our work in that we show the accuracy in the estimated epipolar geometry by calculating a fundamental matrix under conditions that cause degeneracy.

E. FUNDAMENTAL MATRIX ESTIMATION

The epipolar geometry presents the relationship between two rigid scenes. It is defined by a fundamental matrix so the quality of the algorithm used to estimate the fundamental matrix is directly correlated with the accuracy in the epipolar geometry. Hence, Armangué et al, [20] did an analysis on the different methods for calculating the fundamental matrix and used the resulting estimate to measure the tracked features against their respective epipolar lines the same way we do here in this thesis. They evaluated numerous algorithms classified as linear, iterative, and robust. Their conclusion is that linear algorithms are useful as long as there are not a lot of outliers, but in the case that outliers are expected or cannot be dealt with, the LMedS algorithm with M-Estimators should be used.

Although a very similar experiment to the one we conduct described in experiment two, the primary difference between their experiments and ours is that their projection of the points is generalized where our focus is on specifying different camera motions between projections. This specificity allows for pinpointing the component of the camera's motion that affects the quality of the fundamental matrix's estimation of the epipolar geometry under unconstrained camera motion. [20]

F. STRUCTURE FROM MOTION (SFM)

Typically SFM algorithms are an offline, iterative approach to 3D reconstruction that iteratively traverses multiple images in order to minimize the errors of a reconstruction over the entire sequence.

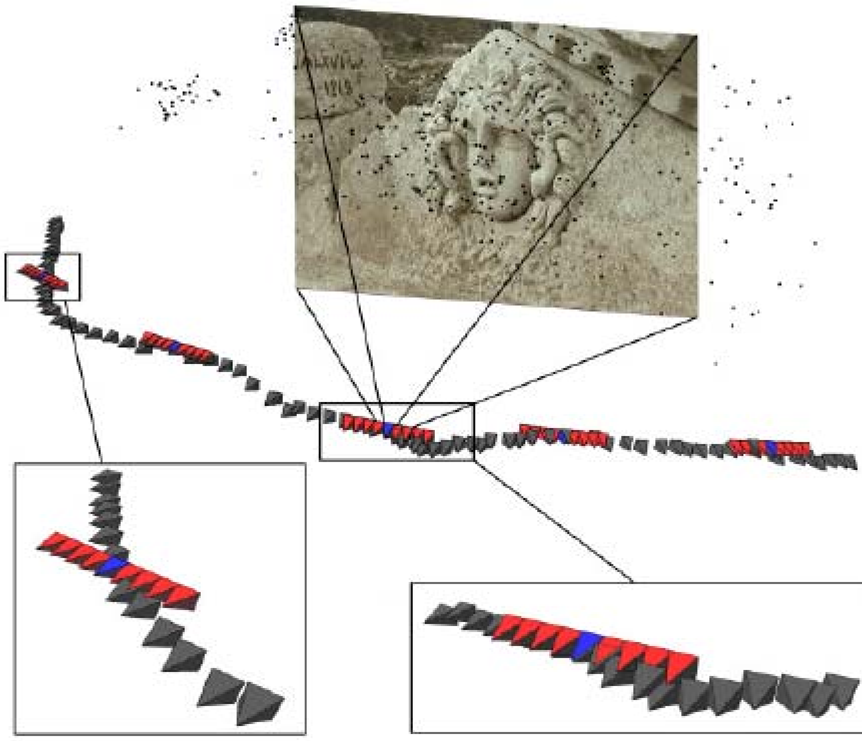


Figure 12. Medusa Recreated Using Structure From Motion.

Multi-view synthesis using SfM and Image Based Rendering (IBR); gray: original camera path, red: virtual stereo cameras, blue: original camera of a multi-view camera setup. From: [21]

In Figure 12, a 3D reconstruction of an ancient Medusa head is illustrated. The camera locations are shown as pyramids, and remnants of the tracked features used in the reconstruction can be seen as dots near the 3D reconstruction itself. The original reconstruction was done by Marc Pollefeys. [22]

According to Jebara et al. [23], video poses a problem to traditional algorithms for SFM due to the small baselines that exist between sequential frames. In order to compensate for this they suggest using information over numerous frames. This is very similar to the method for three-dimensional reconstruction from uncalibrated views that use multiple scenes to iteratively triangulate three-dimensional points from the multiple two-dimensional projections, which uses a Levenberg-Marquardt non-linear optimization. [12]

This relates to our overall goal in that it provides the method for which the images can be processed in order to efficiently create a virtual model from the robot's video stream.

G. LEARNING ALGORITHMS

Researchers at Stanford have built a training set (found at http://ai.stanford.edu/~asaxena/learningdepth/data.html) from monocular images and ground truth laser range images from a SICK laser. The training set was used to train a Markov Random Field (MRF) to estimate depth for unstructured scenes. They have two approaches. The first uses a Gaussian MRF, the second a Laplacian MRF. The approaches both focus on the use of scene texture to cluster patches of the scene together. That is, two patches that both contain similar colors from the side of a building and are next to each other will have highly correlated depths in the overall scene. The Laplacian approach proved better due to its ability to deal with sharp edges and was more robust to outliers. They successfully used the algorithm on an r/c car which navigated through an outdoor environment at high speeds. Figure 13 consists of results from the experiment showing the original image, the ground truth, and the Laplacian estimate. [24]

Their work is related to this thesis in that they are able to create a dense range image of a scene using a monocular camera. However, their work is very recent and does not follow the traditional approach of SFM algorithms using the epipolar geometry between images. .

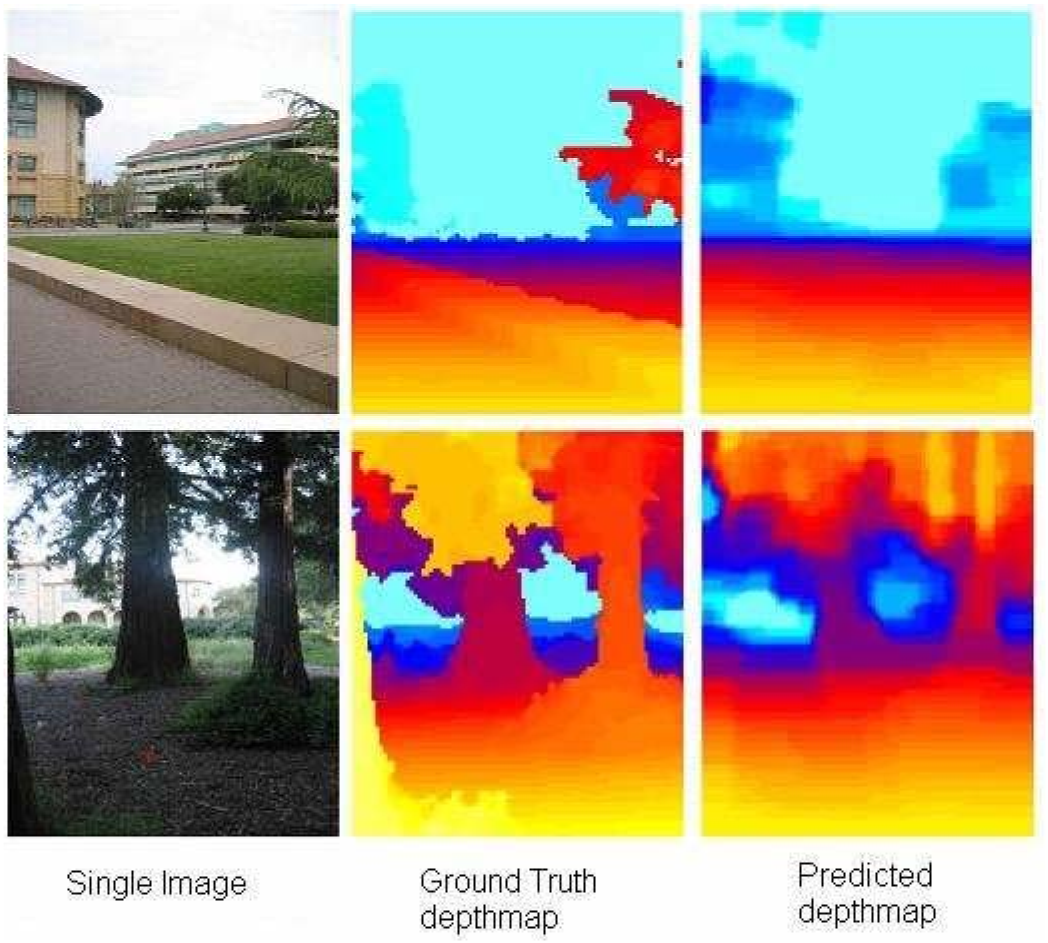


Figure 13. Markov Random Field's depth estimation for two different scenes.
From: [24]

Another difference in their approach is that the MRF must be trained on an environment before the algorithm is useful where an epipolar geometry exists between any two images.

H. STRUCTURE FROM OCCLUSION

Another novel method for recovering the third dimension from images is using occlusion. In Figure 14, the actor is weaving in and out of the columns and as he does so a Bayesian based sensor fusion algorithm processes the cues and can recreate the original scene

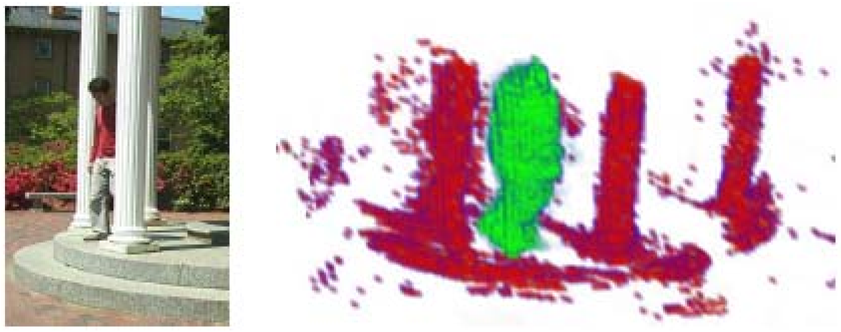


Figure 14. 3D Reconstruction Occlusion Inference.

Left: A frame from the original video sequence.

Right: A 3D reconstruction of the pillars created as the actor weaves in an out of the columns. From: [25]

By learning the background, the dynamic subject can be automatically removed from the image using standard computer vision algorithms for silhouette extraction. The general technique of recovering a model of the scene is Shape-from-Silhouette (SfS). When the silhouette, visual hull, is partially occluded, then inference can be made as to the ordering in the scene of pixels in relation to the camera view. [25]

This approach is beyond the scope of this thesis because of its reliance on repetitive motion of a dynamic object with a static camera, where our focus is on the use of a mobile camera. Furthermore, it does not directly give information on the Euclidean structure related to the rest of the scene.

I. ALGORITHM COMPARISON

We look at primarily three different algorithms as well as a few robust versions of them. They are outlined below along with a section detailing what differences between them cause the differences in the results of our experiments.

1. Hartley and Zisserman Based Implementation

The first algorithm explored is our own implementation of the 8-Point algorithm as described in [1] and first proposed in [26]. Although fully explained earlier in this chapter, in brief this algorithm linearly estimates the fundamental matrix by creating a single matrix, *A,* where each of the rows is the Kronecker product of the normalized corresponding points and the fundamental matrix is the null space of *A*. If the corresponding points are $\mathbf{x}_i$ = [*x*, *y*, 1] and $\mathbf{x}_i'$=[*x*', *y*', 1] then the calculations are as follows:

$$x_i' x_i f_{11} + x_i' y_i f_{12} + x_i' f_{13} + y_i' x_i f_{21} + y_i' y_i f_{22} + y_i' f_{23} + x_i f_{31} + y_i f_{32} + f_{33} = 0$$

$$\left(x_i' x_i, x_i' y_i, x_i', y_i' x_i, y_i' y_i, y_i', x_i, y_i, 1 \right) \bar{f} = 0$$

$$A\bar{f} = \begin{bmatrix} x_1' x_1 & x_1' y_1 & x_1' & y_1' x_1 & y_1' y_1 & y_1' & x_1 & y_1 & 1 \\ x_2' x_2 & x_2' y_2 & x_2' & y_2' x_2 & y_2' y_2 & y_2' & x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_8' x_8 & x_8' y_8 & x_8' & y_8' x_8 & y_8' y_8 & y_8' & x_8 & y_8 & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix} = \bar{0} \qquad [1]$$

Here, the variables f_{ij} are the unknown entries of the fundamental matrix and $A\mathbf{f}$=0 is either over- or underdetermined based on the number of corresponding points being used. ***f*** is the null space of A and can be solved with a Direct Linear Transformation using Singular Value Decomposition (SVD). [1]

2. Structure and Motion Toolkit in Matlab

Philip H. S. Torr, while at Microsoft Research, created a Structure and Motion Toolkit (SAM) implemented in Matlab. SAM comes with a tool (*torr_tool.m*) that easily demonstrates how the process proceeds from feature tracking through determination of structure and motion. It is a GUI based interface that allows the loading of two images. Once loaded, the algorithm performs feature matching between them using a rough implementation of a Harris corner detector. From the detected features one can determine correspondences between the images which is used as input into an estimation of the fundamental matrix. Finally, the structure and motion of the images will be derived from the triangulation between the images and displayed. SAM also has with it an 85 page manual briefly describing the various tools available and the theories behind them. The toolkit is based upon his PhD thesis, which, a good reference for more in-depth understanding. [27]

Although this toolkit at first appears to be all encompassing, it is better used as a starting point rather than be relied upon in its current state. There are various, hard coded parameters in the programming so when using any of the provided Matlab files, the code should be reviewed to make sure it performs as advertised.

The primary matlab file (.m file) used for estimating the fundamental matrix in the toolkit is *torr_estimatef.m*. It includes a linear calculation that is very similar to the method found in [1]. The same file also allows for more robust estimations of the fundamental matrix. Most notable are the Maximum A Posteriori SAmple Consensus (MAPSAC), and Maximum Likelihood Estimation SAmple Consensus (MLESAC). These robust estimators are presented to deal with outliers in the tracked features. For this experiment, the intent is to determine how well these estimators can estimate the fundamental matrix by, again, measuring the error in the resulting epipolar geometry. [28]

In brief, MLESAC is a generalized version of RANdom SAmple Consensus (RANSAC) and was first proposed in [29]. Where RANSAC is a "hypothesis and test" algorithm declaring a solution when a certain threshold is obtained, MLESAC measures the quality of a robust solution with the log likelihood. [9]

Our experimentation with the toolkit is limited to the use of three algorithms: linear, MAPSAC, MLESAC. There are other estimators in SAM that can be explored as well in the future, but they are incomplete in their implementation.

3. Intel's OpenCV Implementation

Intel's OpenCV (Open Computer Vision) library makes available four different methods for computing the Fundamental Matrix from corresponding points: 7-Point, 8-Point, RANSAC, and Least MEDian of Squares (LMEDS). We tested the RANSAC, LMEDS and 8-Point algorithms in this thesis. The details are described in the next section.

4. Primary Differences

The heart of the 8-Point Algorithm for calculating the fundamental matrix is the *A* matrix. Each row of the A matrix is the Kronecker tensor product of the corresponding points between two images where the point in the second image is multiplied by its corresponding point in the first image (as opposed to the points in the first image with those in the second which gives a different result). The only input into algorithms for calculating the fundamental matrix that we are evaluating we corresponding points between two images. The first distinguishing difference between the algorithms is how these corresponding points are normalized prior to creating the *A* matrix. Ma's algorithm uses Cholesky factorization to create the normalizing transformation for each image such that

$$K = chol\left(\frac{x * x^T}{\# Cols}\right)^{-1}$$

where K is the normalizing transformation, and x is the homogenous Cartesian coordinate of a feature in the image. The H&Z algorithm creates a normalizing function T for each image such that when multiplied by a point, the average distance of each point from the that point to the origin is the $\sqrt{2}$.

$$scale = \frac{\sqrt{2}}{\frac{\sum_{i=1}^{n}\left(\sqrt{\left(x_i - \overline{x}\right)^2 + \left(y_i - \overline{y}\right)^2}\right)}{n}}$$

$$T = \begin{bmatrix} scale & 0 & -scale \times \overline{x} \\ 0 & scale & -scale \times \overline{y} \\ 0 & 0 & 1 \end{bmatrix}$$

where $\overline{x}, \overline{y}$ are the average x and y values for the set of points in a single image. Torr's Structure and Motion library takes an alternative approach to normalizing. The Ma and

H&Z algorithms assume the homogenous value for the vector ***x*** is one. Torr assumes that it is the middle of the x axis of the image space. For instance, if the original image is 512x512, then the homogenous value is 256. When *A* is calculated with the Kronecker tensor product, the columns whose values are the product that includes the homogenous value, then they are significantly scaled by this larger value. For instance, the last column of *A* becomes 256*256 versus 1*1 for Ma and H&Z. The effect should be the same but the difference is that the 256 scales the normalized points in such a way that they are closer to their initial value versus the effect of the homogenous value of one.

OpenCV's 8-Point algorithm uses the same algorithm as H&Z for calculating a normalizing transformation. The primary difference between the two algorithms is the implementation language. H&Z is implemented in Matlab, where OpenCV is implemented in C/C++. Monitoring the progression of the two algorithms, the two return the same results until the SVD of the A matrix is found. If $[u \;\; d \;\; v]$= SVD(*A*). The u and d matrices are the same (left orthogonal, and singular values matrices). The difference is in the last two columns of v^T matrix. Since the entries of the normalized Fundamental matrix comes from the last column of the 'v^T' matrix in column major order. The major difference in the algorithms comes from the implementation of the SVD in OpenCV. In order to rectify this difference we used different SVD implementation from CLApack, LApack, and even Matlab's SVD compiled in C. All three libraries returned different results than Matlab's SVD running natively. This difference is significant enough that the algorithms are deemed to be different in regards to our experimentation.

The last difference worth noting is that of RANSAC in OpenCV. Its candidate Fundamental matrices comes from selecting seven points (versus eight or more for the 8-Point algorithm) and running the 7-Point algorithm on these points. Once the best fundamental matrix is found, or n number of iterations have been completed, the algorithm continues on to calculate a fundamental matrix from all inliers using the 8-Point algorithm (assuming there are at least 8 points available and a flag is set to use the 8-Point algorithm when RANSAC is called).

IV. EXPERIMENT: INTRODUCTION

A. OVERVIEW

The overarching goal of our experimentation is to determine how well various algorithms perform under different camera motions, including those motions that are degenerative such as pure rotation. The experimentation sequentially progresses through the four different stages of reconstruction: feature detection and tracking, initial estimation of the fundamental matrix, non-linear refinement of the estimation, and Projective to Euclidean geometry transformation.

Each experiment has four sections. The first section is a description of the overall experiment which is closely related to the second section that details our method. The third section is filled with either all the results, or a partial set of results, and where the results came from. The last section is a discussion describing the outcome of the results. Due to the sheer amount of data produced by some experiments, the complete results may reside in an Appendix. When possible, however, we present the results along with the description for ease of reference.

Our first experiment relates to the first stage of 3D reconstruction; feature tracking. Although not specifically targeting the topic of the thesis as it relates to camera motion, feature tracking is the essential first step for vision-based SLAM. During our research we happened across a novel method of performing feature tracking which is introduced in the experiment. Essentially this algorithm moves the robust operation of delineating between inliers and outliers to the operation of feature tracking itself and reduces the noise in the point correspondences between images that are passed onto downstream algorithms. That is, it improves the feature tracking operation of current algorithms making the point correspondences found in real world images higher quality which leads to better results for downstream operations.

The second experiment focuses on camera motion and looks at the quality of the epipolar geometry derived under different camera motions for different algorithms in early stages of a projective reconstruction.

The third experiment is very similar to the second, but utilizes a different measure of quality and measures it late in the projective reconstruction process.

The fourth experiment looks at the quality of estimates of the epipolar geometry before and after an iterative update process is used in an attempt to improve the results.

The fifth and sixth experiments look at methods that use known ground truth along with a projective reconstruction to determine information about the location of the camera in a Euclidean space.

B. DEFINING THE SYNTHETIC 3-D SCENES

To control the input for the algorithms, two 3-D synthetic scenes were engineered for use in experiments one through four. The first, shown in Figure 15, is a set of points that form an elongated cube, where the front plane of the cube is nearest the camera and the elongated portion stretches towards the plane at infinity giving projections more perspective. The locations of the front and back planes of the cube allow for the maximum amount of parallax between the two ends and for a more realistic representation of points in a scene. For Ma's algorithm[12], a ninth point is added to this scene between two corner points on the elongated axis of the cube to satisfy its algorithmic requirements. This creates a point that is between the front and back of the cube and should not affect the results enough to change our comparison of algorithms.

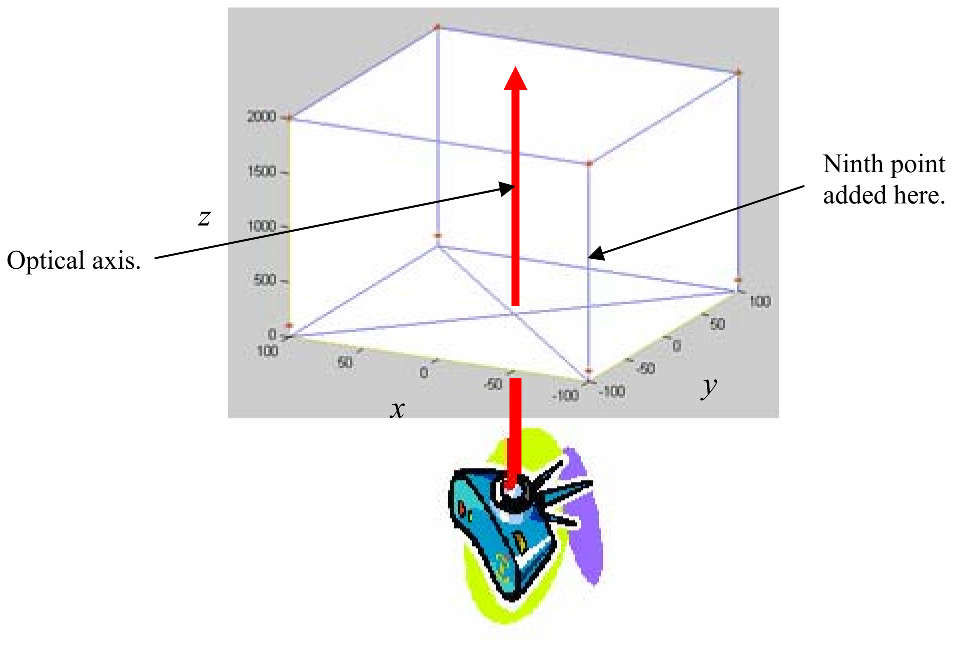


Figure 15. 3D Point Cube.

This is an illustration of the elongated point cube (the red *'s in the image above) overlaid with a frame structure to highlight the cube shape and show coplanarity of the points.

The second synthetic 3D scene, Figure 16, is a total of 200 points in a point cloud which are uniformly distributed in the *xy*-plane. It is engineered by randomly selecting the *x* and *y* from a uniform distribution between 0 and 1, shifting them about the *x* axis by adding -0.5, then multiplying the result by 100 to radiate them away from the origin into all four quadrants of the *xy*-plane starting at (-100,-100) to (100,100). For the z value, depth, a uniform random number was chosen between 0 and 1 and used to draw a value from the distribution defined by $x^{0.2} = z'$. The points are then normalized using the maximum *z* value and then multiplied by 12,000 in order to stretch the points out towards the plane at infinity. The last transformation to the point cloud is to shift the points away from the *xy*- plane, and thus the camera, by 500. This allows us to move the camera along the optical axis keeping all points in view. The result is a set of points which is uniformly distributed when perceived in the *xy*-plane, and whose *z* values are skewed towards the camera, with the deepest points found near the plane at infinity. The reason we chose the dimension 200x200 for the xy-plane is to limit the width of the field of view, while at the same time, the depth of 12,000 was such that the points with a depth nearest 12,000 translated very little between projections meaning they were at or near the plane at infinity. At the same time, as the depth of points diminished from 12,000 to zero, the points translated more between projections.

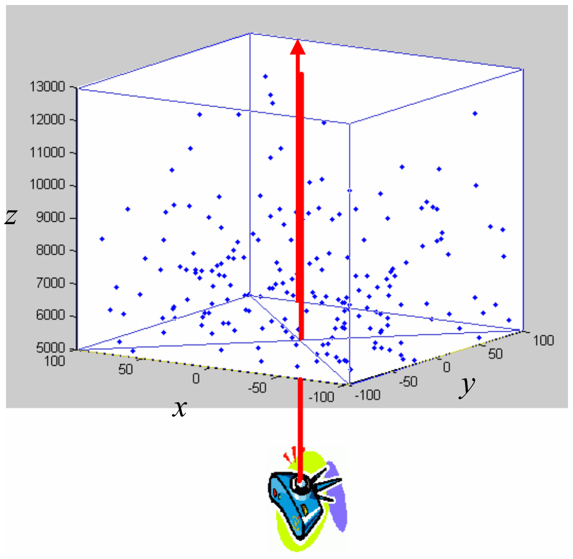


Figure 16. 3D Point Cloud.

A 3-D view of the point cloud of two hundred points centered around the origin. Perspective projections of this cloud are used as the input into the various fundamental matrix estimation algorithms.

The projection of the point cloud provides 200 point correspondences to the algorithms for estimating the fundamental matrix. When the points are perturbed by noise, the error in the resulting epipolar geometry increases. The delta in the baseline for perfect points and those with noise establishes how well the algorithm can deal with various degrees of noise in its input.

In order to establish a baseline of performance, the datasets are projected to create two separate sets of projections. The first set is a direct projection of the points to create perfect point correspondences between the images. This is the ideal situation for these algorithms and provides a perfect environment for accurately determining the epipolar geometry.

The second set of projections is that of a perturbed dataset. To accomplish this, each entry (x,y,z) of a point is multiplied by a random number, each generated from a Gaussian distribution with a mean of 0 and a standard deviation of 0.025.[1] The product is then replaces the original entry to introduce various offsets in the x, y, and z directions.[2] For example, if the original *x* value is 5, and the random number selected is 0.025, then the final x value (x_{new}) will be: $x_{new} = (5 \times 0.025) + 5 = 5.125$.

The overall resulting effect is the addition of noise to the original points such that 60% of the points will have their coordinates altered by less than 2.5% and 95% of the original points will have their individual coordinates altered by less than 5%. The points perturbed the most are equivalent to either points being poorly tracked or superfluous points.

Why do we use these two datasets with the various projections? The answer is that we have generalized the problem of feature tracking to two possible cases. The first is that we are in a feature rich environment where we can acquire a large number of point correspondences. The second case is the opposite in that we are in a relatively featureless environment and can only find and track a few data points that may not be optimally positioned (i.e., tracked features are co-planar). Using these two cases as a guideline, we then derived our datasets. Note; however, that once these datasets are established, both with and without noise for a total of four synthetic scenes, they remain static and reused for all experiments. This ensures that it was the camera motion coupled with the algorithm that influenced the error in the epipolar geometry, versus having chosen a good or bad dataset for the particular trial.

[1] The choice to use the distribution with a standard deviation of 0.025 is from the idea that errors are relatively small for a majority of the points and larger for a few. This is simply an educated guess at a representation of real world due to the fact that there is frequently errors in feature tracking.

[2] At first glance, the introduction of noise on the *z* value may seem to have no effect, but due to the perspective projections being created, a change in the *z* value will add noise to the *x,y* location of the resulting projection.

C. CAMERA MOTIONS

In order to produce conditions that provide decidable results we leverage both Ma's[12] and Hartley's[1] comments that lead us to believe that the amount of motion parallax between images can play a role in estimating the epipolar geometry. By controlling the camera motion between projections we can control how much parallax will exist between projections and hence test each algorithm’s ability to estimate the epipolar geometry under these different conditions. If the camera is moving between projections the tracked features will change their location, i.e., translate in the *xy*-plane. When comparing all translation for all points in the images, we identify three different types of relative motions features can move between projections.

1. An equal amount of translation for all points.

2. Different levels of translation in the same direction.

3. Different levels of translation but in different directions.

Each of these correlate directly to three different camera motions

1. Rotation about the camera’s y-*axis* (camera’s vertical axis).
2. Pure translational movement in the xy-plane (camera plane).
3. Pure translational movement along the *z-axis* (camera’s optical axis).

For each algorithm we will create a pair of projections of each of the three-dimensional datasets for each camera motion above, as well as unconstrained camera motion. This means each algorithm will have four different sets of projections as described by the constraints outlines above and detailed below. We will then evaluate the error in the epipolar geometry for each pair of projections and record this in a table in Appendix A, and discuss the results in the next chapter.

1. Rotation Around the Camera’s Z-Axis Motion Constraint

With this motion constraint we attempt to remove all parallax from the perfectly matched point projections and leave only erroneous parallax for the noisy projections. To do this, the camera is rotated about its vertical (*y*) axis between projections. The importance of this test is to determine if the algorithms can return suitable estimates of the Fundamental matrix even though the algorithm may not be well conditioned as described by Ma et. al. [12] The actual implementation for this was an initial projection of the synthetic scene onto the image plane, then a translation of these projected points by an equal amount in the x direction. This is equivalent to rotating the camera because there is no parallax in the image due to an equal translation of all points.

2. Translation Along the X Axis Motion Constraint

In order to create a pure translational motion of the camera plane, we alter the camera's *x* value between projections as it relates to its world coordinate position (*x,y,z*). The effect of this motion is that parallax between the projections is maximized. For illustration purposes, a resulting projection of the noiseless point cube is shown in Figure 17. In the images, one point is occluded but is still used in our experiments.

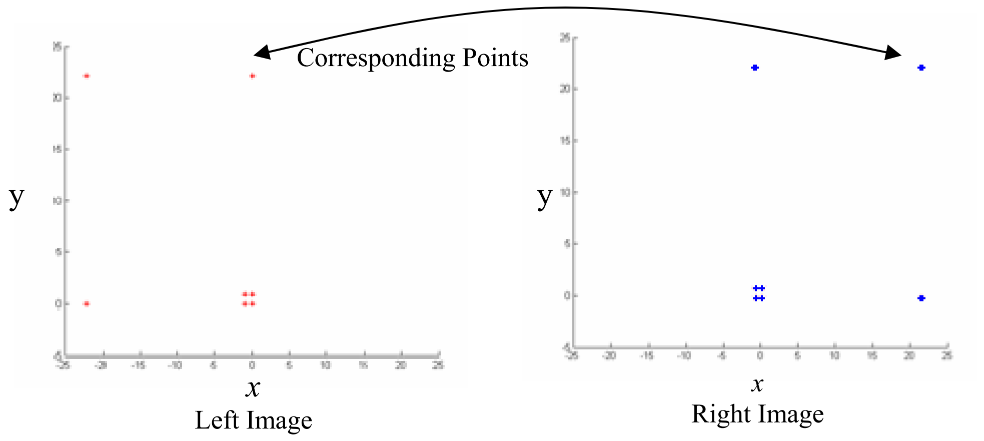


Figure 17. Projection of Point Cube Under Pure Translation.

Shows the projection of the elongated point cube. The large perceived square has the same (*x,y*) coordinates as the small square. The only difference is their *z* (depth) values. Due to the positioning of the camera the bottom-right most corner point of the small square is occluded by the corner of the large square so only 7 points appear in the image.

3. Translation Along the Z-Axis (Optical Axis) Motion Constraint (Referred to as Scaling)

For this constraint, the camera's pose remains constant except for its position in the world coordinate frame along the *z*-axis. The resulting effect is that the points radiate outward from the optical axis, which is also the center of the image. Under this motion constraint the relative movement of the points between the foreground and background is based upon the depth of the point. If a point is on or near the plane at infinity then it will see little change in its projected (*x,y*) values between translations. Conversely, points closest to the camera will remain in front of the camera due to their initial distance from it, but will have noticeable translation outward away from the optical axis.

4. Unconstrained Motion

The last pair of projections is created by unconstrained movement of the camera which contains components of all other constrained motions. Between projections the only pose values that remain constant is the pitch and roll. The reason for exclusion of these camera motions is that a change in pitch is the same as a change in yaw just along a different axis, and a change in roll has the same effect of zero parallax shift between projections as a rotation. All other freedom of movement is explored by rotating the camera about its vertical axis, then translating the camera. This allows us to explore what effect each of the different components of motion has on the unconstrained motion which is most similar to what is experienced in the real world.

D. BASELINE ESTABLISHMENT

The ground truth error for any epipolar geometry is that the points will lie on their respective epipolar line making the total error in the epipolar geometry zero. However, to establish if an algorithm as presented can actually predict this ground truth, we establish a baseline of performance. That is, we present each algorithm with projections of both the cloud and cube datasets made up of perfect points meaning they are perfectly matched (one-to-one correspondences without outliers) and are perfectly placed (no deviation in their *x* and *y* position from noise). Given these conditions, an algorithm should be able to predict ground truth, and any failure in doing so is established in the performance baseline.

When the datasets are then projected with noise, the performance of the algorithms can be determined by comparison to their respective baselines versus a comparison to the ground truth which it may not be able to predict in the first place

E. EXPECTED RESULTS

Based upon Ma's comments on parallax, our expectation before running our experiments is that the algorithms will best predict the epipolar geometry when the camera motion experiences pure translational movement due to the high amount of parallax. Along the same lines we expect pure rotational motion, which lacks parallax, to give us the most error and that unconstrained movement will lie somewhere between the two due to its translation and rotation components. When the points are perturbed with noise, the expectation is that the ability of the algorithm to predict the underlying epipolar geometry will be diminished.

THIS PAGE INTENTIONALLY LEFT BLANK

V. EXPERIMENT 1: MULTI-PASS FEATURE TRACKING

A. DESCRIPTION

The most well known reconstruction algorithms begin by acquiring an image and determining which features, if any, can be detected and tracked from image to image. The tracked feature locations are what the reconstruction algorithms require for their determination of structure and motion.

The standard method for processing images is to acquire one, search it for quality features, and then try to find those same features in the subsequential image. This method essentially limits the amount of processing time that is spent on any individual image as a whole and quickly reduces the problem to sets of (x,y) pairs that are later classified as inliers and outliers. This process is sequential and the movement always forward in the sequence.

This experiment proposes the idea of moving both forward and backward in the stream of frames to verify if we are accurately tracking features between frames. This method of double checking feature tracking results could possibly reduce the number of outliers that are passed to downstream processes. With regards to traditional robust methods for dealing with outliers, this could prove to be either a supplement or complement.

We call this method multi-pass feature tracking and it is conducted by a repetitive search of the same frames by iterating between frames n and n-1 until a defined condition is met at which time the proposed algorithm moves on to repeat the process in frames n and n+1. A graphical depiction is in Figure 18.

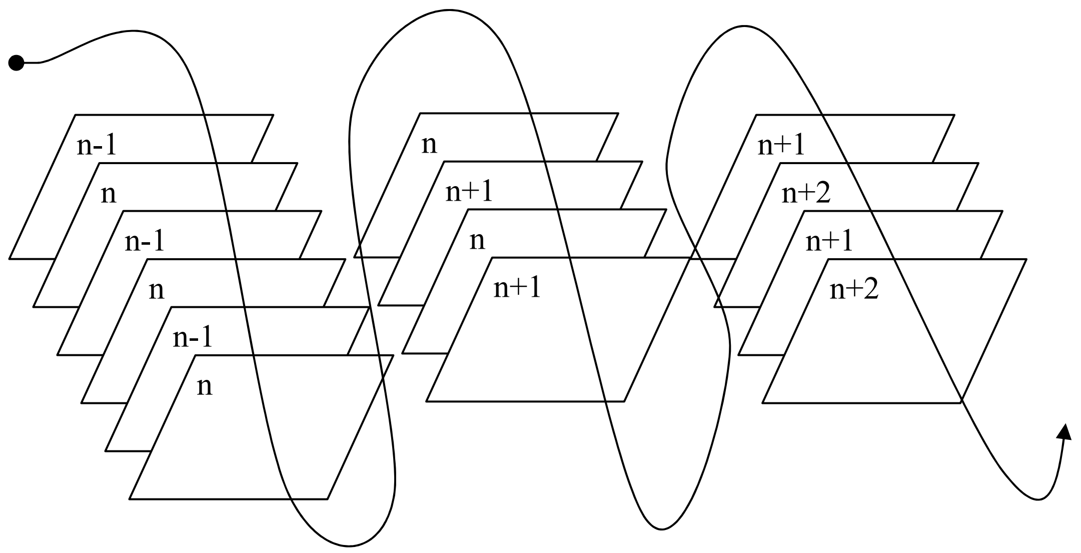


Figure 18. Multi-Pass Feature Tracking Illustrated.

Depiction of a possible flow of multi-pass feature tracking using an exit criterion that ends the iterative process when the number of points tracked remains static. In fact, each stack of frames is repetitively searched until at least one of two conditions is met: (1) stable number of features (convergence), or (2) a maximum number of swaps between frames (threshold).

The next downstream process in reconstruction algorithms from feature tracking is an initial estimate of the fundamental matrix, which defines the epipolar geometry between two images. Outliers have an adverse effect on this process, so there are robust methods for dealing with outliers during the estimation of the fundamental matrix which were described in Chapter II. Our multi-pass feature tracking approach is directly testable by comparing it to current robust methods, and in our case we will use RANSAC as the current standard. Using the error measurement also described in Chapter II, we can compare the errors in an epipolar geometry calculated using our algorithm compared to the errors in the epipolar geometry estimated from RANSAC. If our algorithm results in less overall error in the estimate of the epipolar geometry then our method is at least comparable to current robust estimators.

B. METHOD

This experiment begins by utilizing the tools available in Intel's OpenCV library to both track features in real images, as well as to calculate the resulting fundamental matrix. The optical flow is calculated using cvCalcOpticalFlowPyrLK which is a pyramidal implementation of an algorithm introduced by Lucas-Kanade. [8]. We implement the function using a three level pyramid.

The features are tracked in two different sets of images. The first is a capture of a three-dimensional, virtual scene with roughly translational movement of the camera. The second set of images is a fruit bowl provided by Philip Torr's Structure and Motion demonstration which has mostly rotational movement of the camera. [27]

The estimated fundamental matrix (*F*) between the scenes is found using OpenCV's cvFindFundamentalMat. This method is called using either the linear 8-Point (linear) option, or the RANSAC[3] (robust) option. The quality of the algorithm is measured by the error for each point in the resulting epipolar geometry, defined as $x'^{T}Fx=error$, such that x and x' are corresponding points and F is the fundamental matrix. This method of measuring error is, again, detailed further in Chapter II.

The control for the experiment is established by measuring the error from a single-pass feature tracking approach with both a linear and robust estimation of *F*. The alternative multi-pass approach is then explored by using the same sets of images. There are three different configurations for testing the multi-pass approach. The first is to process the features found after *n* different swaps between the frames, where *n* is a fixed threshold for the number of times the frames are swapped. *n* can be any number greater than one. The downside of this approach is the case where a set of images have features being tracked perfectly between them. There can be an unnecessary cost in processing time for each pair of frames.

[3] Note that in OpenCV, RANSAC is a robust version of the 8-Point call in that it utilizes the 8-Point algorithm to calculate the fundamental matrix, but does it through a hypothesis and test approach.

The second option for processing images is to exit the iterative process once the number of features in the image stabilizes for *m* frames. For our purposes we set *m* equal to two (*m*=2) which causes the algorithm to continue if the number of features tracked between frames *n* and *n*+1 are equal, i.e., the number of tracked features being between the *m* images converge.

The third, and final method tested is a combination of the first two. There is an upper limit to the number of swaps that occur for any two frames set by a threshold, coupled with the possibility that the threshold is not reached every time due to stability in the number of tracked features. Below, in Figure 19, are the two sets of images used.

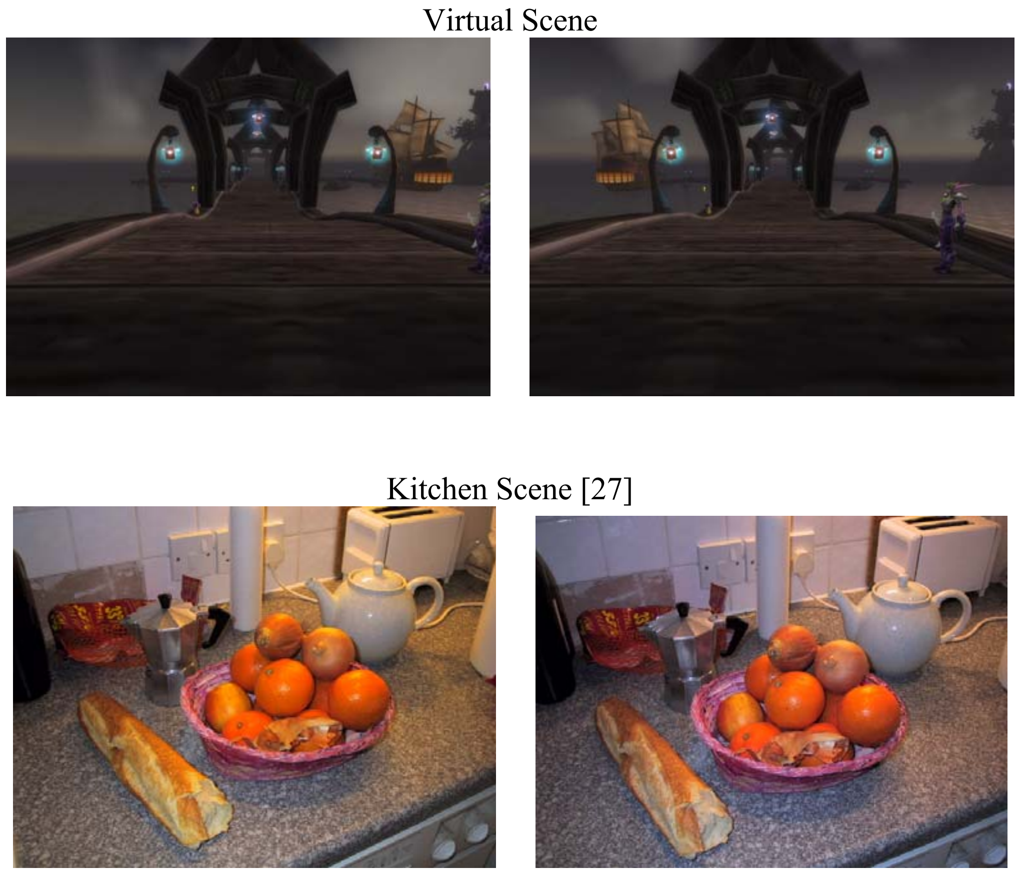


Figure 19.   Images used in Multi-Pass Experiment.

Image pairs used for testing single-pass versus multi-pass feature tracking.

C. RESULTS

Tables 2 and 3 (on page 67 and 68) summarize our results of the experiment and are categorized for each of the images, Virtual scene and Kitchen scene respectively. We decided that the measure of error we would use is average of the absolute (L1) distance errors and its associated standard deviation. Hence, the columns are as follows:

1. Algorithm: refers to the method of feature tracking used as well as the method used for calculating the fundamental matrix.
2. Iteration Exit Condition: describes the point at which the multi-pass algorithm exits the iterative process of frame swapping between two frames.
3. Average of the Absolutes: because a measured distance from the epipolar line in one direction is equivalent to the distance measured from the epipolar line in the opposite direction, this column shows the average of the absolute distances.
4. Standard Deviation: the standard deviation of the average of the absolute distances.

The multi-pass approach to feature tracking resulted in improved linear estimates of the epipolar geometry for both the virtual and kitchen scenes. When using the linear 8-Point algorithm with the tracked features of the virtual scene, the traditional single-pass approach had a distribution of errors 0.1226 ± 0.0695 versus multi-pass's 0.0781 ± 0.0595 when using the convergence criterion and 0.0753 ± 0.0486 when using straight threshold criterion. Accordingly, using the features tracked for the kitchen scene with the same algorithm, we found single-pass's distribution to be 0.1230 ± 0.1054 versus multi-pass's 0.0793 ± 0.0591 when using a convergence exit criterion, and 0.0752 ± 0.0471 for a straight threshold criterion.

In an attempt to improve upon multi-pass's results, we coupled it with RANSAC and compared the results to that of the traditional single-pass approach using RANSAC. In general, single-pass with RANSAC had errors less than multi-pass approach and

RANSAC, at least for three of the four trials. The only trial where multi-pass showed better performance was with the virtual scene when it used a straight threshold exit criterion, resulting in 3.7133±4.9976 versus single-pass's 12.5123±27.2681. In comparison, the results for the robust estimation algorithm with single-pass for the kitchen scene, were 2.6222±5.6178, versus multi-pass's 4.8345±2.9335 with a convergence criterion, and an even worse 5.4606±3.4453 using a straight threshold criterion.

Lastly, the best performance achieved in this experiment came from the multi-pass approach when using a straight threshold exit criterion, and passing the resulting point correspondences to the 8-Point algorithm. For both scenes, this approach had identical distributions when rounded of 0.08±0.05. Note; however, that it was only slightly better than the use of a convergence exit criterion (0.08±0.06) which stabilized in the number of points it was tracking only after three image swaps.

Algorithm	Iteration Exit Condition	Average Of Absolutes	Standard Deviation
Single Pass w/ 8-Point	n/a	0.1226	0.0695
Single Pass w/ RANSAC	n/a	12.5123	27.2681
Multi-Pass w/ 8-Point	On Convergence or Threshold	0.0781	0.0595
Multi-Pass w/ 8-Point	Threshold Only	0.0753	0.0486
Multi-Pass w/ RANSAC	On Convergence or Threshold	15.3970	28.9191
Multi-Pass w/ RANSAC	Threshold Only	3.7133	4.9976

Table 2. Errors in Epipolar Geometry for the Virtual Scene.

The error in the epipolar geometry for different approaches to feature tracking: single pass versus multi-pass. "Iteration Exit Condition" is the condition under which the multi-pass algorithm considers that all features have been tracked forwards and backwards successfully "Convergence" means that the number of points being tracked between frames stabilized. The error is defined by $x'^{T}Fx = error$, which determines the error in the epipolar geometry where the variables x' and x are corresponding points between the two images and F is the fundamental matrix.

Algorithm	Iteration Exit Condition	Average Of Absolutes	Standard Deviation
Single Pass w/ 8-Point	n/a	0.1230	0.1054
Single Pass w/ RANSAC	n/a	2.6222	5.6178
Multi-Pass w/ 8-Point	On Convergence or Threshold	0.0793	0.0591
Multi-Pass w/ 8-Point	Threshold Only	0.0752	0.0471
Multi-Pass w/ RANSAC	On Convergence or Threshold	4.8345	2.9335
Multi-Pass w/ RANSAC	Threshold Only	5.4606	3.4453

Table 3. Errors in Epipolar Geometry for the Kitchen Scene.

The error in the epipolar geometry for different approaches to feature tracking: single pass versus multi-pass. "Iteration Exit Condition" is the condition under which the multi-pass algorithm considers that all features have been tracked forwards and backwards successfully Where "convergence" means all points were tracked between the frames. The error is defined by $x'^T Fx = error$, which determines the error in the epipolar geometry. The variables x' and x are corresponding points between the two images and F is the fundamental matrix.

D. DISCUSSION

When comparing the single-pass approach to the multi-pass approach, we find that a linear algorithm using point correspondences derived from the multi-pass feature tracking approach is better able to estimate the epipolar geometry in a scene. In our experiment we use the robust method RANSAC because it uses a linear estimator of inliers to get a final result. Its classification of inliers is equivalent to multi-pass's exclusion of points that are poorly tracked, hence we can directly compare the two as robust approaches. Therefore, as a robust method, the multi-pass shows better performance than RANSAC for both the Kitchen scene and the 3-D Virtual Scene. This being the case, the multi-pass approach appears to be a supplement to traditional robust methods. The cost of using a multi-pass approach versus single-pass is that each frame has to be processed multiple times versus RANSAC which is iterative as well, but performs its iterations only after the problem is reduced to (x,y) coordinates, and only does so on a subsample. Therefore, the multi-pass approach increases the computation time for feature tracking and lowers the number of frames per second the overall algorithm can process especially as the features being tracked become more complex, i.e., SIFT or SURF. We believe this to be more of a hardware issue than an algorithmic issue and the benefit of higher quality point correspondences may outweigh the increased computation time depending on the situation.

Despite the excellent results from our algorithm, we have since though of improving on our implementation by including memory. We used the Lucas-Kanade optical flow pyramid approach to track the features back and forth, and only compare the current frame to the last frame rather than using memory to compare the where the features are for every frame swap. An improved algorithm would monitor the location of the points from start to end to determine if a feature is walking away from its original location with each swap. These walking features would then be marked as poorly tracked, classifying them as outliers, and would not be passed on to downstream operations. This approach would possibly provide a more robust approach especially when dealing with environments where repetitive patterns are prevalent, such as tiled floors and ceilings.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. EXPERIMENT 2: ERROR IN THE INITIAL ESTIMATE OF THE FUNDAMENTAL MATRIX MEASURED BY ERRORS IN THE ASSOCIATED EPIPOLAR GEOMETRY.

A. DESCRIPTION

We would like to determine the effect camera motion has on the cumulative errors when performing a projective reconstruction, so it is beneficial to know how the camera motion causes errors at different points in the algorithm. Hence, this experiment looks at errors in the epipolar geometry when the fundamental matrix is initially estimated early in a projective reconstruction. Note that due to the relationship between each camera motion and the degree of parallax it induces between projections, the effect camera motion has on the errors is an equivalent to the effect parallax has on the errors.

Specifically, this experiment determines the errors in epipolar geometry, measured as the distance between the 2D Cartesian coordinates of the original point correspondences and their respective epipolar line. As a reminder, for corresponding points between two different images, the point in one image projects an epipolar line in the second image, and the corresponding point should lie on this line. In order to measure this distance we need an estimate of the fundamental matrix. There are multiple algorithms available to find estimates of the fundamental matrix, all very similar, with specific differences described in Chapter III. The similarity between the algorithms is desired because we would like to know that any error is a result of the camera motion, versus being caused by either the algorithm itself or its method of implementation.

B. METHOD

The synthetic scenes will be projected in such a manner as to produce every desired camera motion, as described in Chapter IV. Once projected, the resulting 2D points in the two images are known correspondences and represent tracked features between non-synthetic images. These correspondences are used as input to our algorithms for calculating the fundamental matrix, which defines the epipolar geometry.

Errors are calculated using the equation *error*$=x'^{T}Fx$[4]. Although any individual calculated *error* can be positive or negative, the statistics we use are that of the absolute value of the *error* values. This method is justified because the sign only determines which side of the epipolar line the point lies. That is, it is positive if the point lies above its respective epipolar line, and negative if the reverse is true. Hence, the sign of the *error* is inconsequential to the *error* itself.

Again, the only input to this experiment are the point correspondences. For the fundamental matrix calculation, these correspondences are first normalized, and then used to find the A matrix (see Chapter III). The outputs of the algorithm are the individual *error* values for each point correspondence from the equation *error*$=x'^{T}Fx$. The algorithms used for this experiment were described, and differences compared, in Chapter III. It is possible to determine each algorithm's performance as measured by the amount of error in the resulting epipolar geometry, but what is of interest to this thesis is the influence the camera motion has on calculated errors.

In order to maintain consistency, the point correspondences, both noisy and noiseless, are calculated as described in Chapter IV and stored to be reused for every algorithm. This way the results are directly comparable.

C. RESULTS

The complete set of summary results by algorithm and dataset can be found in Appendix A, Tables 11-42. The first page of Appendix A is a brief primer covering how to read the tables and their organization.

The range of all errors in the epipolar geometry were from a minimum of zero[5] calculated multiple times by different algorithms, to a maximum of 28.5684 which was calculated from the corresponding points projected from the noisy point cloud by OpenCV's robust and linear methods (Tables 30, 32, and 36). As far as each motion, Pure Translation had a minimum of zero, to a maximum of 10.7654, again, calculated by

[4] See Chapter II for more information on this equation.

[5] Zero here is equivalent to any value that has an order of magnitude near zero.

OpenCV's robust and linear methods for the projection of the noisy point cloud. For Pure Rotation, the errors ranged from zero calculated multiple times, to 12.0000 calculated by OpenCV's linear and robust methods from the noiseless projections of the point cloud (Table 29, 33, and 37). For Pure Scaling motion, the minimum was zero calculated using multiple algorithms, to a maximum of 28.5684, which is also the overall maximum found as described above. Lastly, for the Unconstrained camera motion, the minimum was again zero calculated multiple times, to a maximum of 2.0382 for OpenCV's 8-Point and RANSAC methods calculated from the data presented by the projections of the noiseless point cube.(Table 27, and 31)

For the Hartley and Zisserman based algorithm (H&Z), the maximum error 0.2246 was calculated on the noiseless projection of the point cube under Pure Translation camera motion.(Table 11) The minimum was zero found multiple times in H&Z's results.

The use of Ma's algorithm, resulted in a maximum error of 1.0982 found for the in the resulting epipolar geometry of the projections of the noisy point cloud under Unconstrained camera motion.(Table 42) The minimum was zero, again, calculated multiple times for multiple camera motions.

OpenCV's 8-Point and RANSAC algorithms results were exactly the same. LMEDS, on the other hand, was almost the exact same except for its results derived from the noisy projection of the point cube when the camera experienced either Unconstrained motion, or Pure Translation.(Table 28 versus Table 36) It also deviated from the other two for the noiseless projections of the point cube under Unconstrained motions.(Table 27 versus 35) Although they differ for these particular motions, the range of errors for all three OpenCV implementations is the same as the overall range, zero to 28.5684.

In regards to establishing a baseline that is equivalent to ground truth (all errors equal to zero) H&Z's and Ma's algorithms were the only two algorithms to achieve this goal for all camera motions, and only did so for the noiseless projection of the point cloud.(Tables 13 and 41 respectively)

For the noiseless point cube, Ma's algorithm calculated the ground truth for camera motions Pure Rotation, Pure Scaling, and Unconstrained.(Table 39) H&Z was only able to calculate the ground truth values for Pure Rotation, and Pure Scaling, (Table 11) and OpenCV only returned the ground truth error in the epipolar geometry for Pure Scaling camera motion. Although, OpenCV algorithms did not return results for Pure Translation and Pure Rotation camera motions when using projections of the point cube.(Tables 27, 31, and 35)

As far as the noiseless point cloud, OpenCV calculated the ground truth when the camera had motion that was either Pure Scaling or Unconstrained. (Table 33, 35, and 37)

For errors derived from projection of the noisy datasets when compared to the respective algorithm's baseline, H&Z experienced less errors when estimating the epipolar geometry for the projections of the noisy cube than its baseline (Table 12 versus 11). For example, the errors for the Unconstrained camera motion changed from 0.0469 ± 0.0602 for the noiseless projection to $3.5627 \times 10^{-05} \pm 1.6825 \times 10^{-05}$ for the projection of the noisy cube.(Table 11 versus 12) However, the datasets derived from the point cloud saw the reverse effect of adding noise. Errors changed from $9.2440 \times 10^{-16} \pm 1.5878 \times 10^{-15}$ for the baseline to 0.0109 ± 0.0119 for the correspondences perturbed with noise.(Table 13 versus 14).

Under the same guise, for motions between projections of the cube dataset where the baseline of Ma's algorithm predicted the ground truth, the errors increased from an average and standard deviation of zero for the baseline, to a maximum of 0.1612 ± 0.3625.(Tables 39 versus 40) However, where the baseline was not well defined (Pure Translation), the errors decreased from 0.1004 ± 0.1634 for the baseline to 0.0059 ± 0.0080. (Table 39 versus 40) For the point cloud, which Ma's algorithm predicted the ground truth for all camera motions, the errors increased across the board. (Table 41 versus 42)

Note that for OpenCV, the algorithm did not return an estimate of the fundamental matrix for the projections of the point cube when the camera motions were either Pure Translation or Pure Rotation.(Table 27, 31 and 35) However, something curious about the results derived from the projections of the point cube is that the errors under Pure Scaling motion increased from adding noise (zero baseline to 0.2436±0.0163), and the reverse is true for Unconstrained motion (0.3056±0.7046 for the baseline versus zero for the perturbed projections).(Tables 27 versus 28)

D. DISCUSSION

Our goal is to determine if different camera motions, and likewise different parallax, have an effect on the errors in an estimation of an epipolar geometry. There are three possible outcomes:

1. The camera motion has no effect on the errors. Evidence of this would be a lack of a discernable pattern in the results that show consistent effects by a particular camera motion.
2. Two of the motions have correlated effects, and one motion is not correlated with the others. This is derived from the fact that there are three different motions and a binary possibility; estimates or does not estimate the epipolar geometry. Support for this would be a consistent pattern of errors for a set of camera motions and would mean that it is not only the existence of parallax in the image but also the type of parallax that is causing the effect. Although scaling and translation each have parallax effects, the former causes a radiating out of the points from the optical axis, where in the latter all parallax shifts are in the same direction (also discussed in Chapter IV).
3. The effect of scaling and translation motions are the same, but different from that of rotation. This is the special case where it is the existence of parallax between points in images that causes the effect on errors versus no parallax. This finding would support Ma’s claim that translation must be greater than zero.

We will look at each of these possibilities individually and determine if the results presented in the last section support or contradict each of the hypotheses.

With regard to the first possible hypothesis, we found that scaling and rotational camera motions had very consistent outcomes for the errors in the epipolar geometry, where translational motion was evenly split in its results for being able to predict the epipolar geometry. Hence, scaling and rotational motions have a discernable pattern in the results where they generally have positive effect on the errors in the epipolar geometry. This does not support the first hypothesis. However, the lack of a consistent effect in errors from translational motion does support this hypothesis.

For the second hypothesis, the scaling and rotational camera motions were highly correlated in that they both generally had positive effect on the errors in the epipolar geometry. This observation supports this hypothesis. Furthermore, the inability to decidable discern a pattern in the errors caused under translational motion could also mean that for the type of parallax shift caused by translational motion could have no effect on the estimation of an epipolar geometry.

Lastly, for the third hypothesis, the lack of correlation between scaling and translation means there is no support for this hypothesis in our results.

What is of most interest are the results of the noisy projections of the point cloud using unconstrained motion, which has the most well established baseline. It also is much more similar to real world point correspondences due to the fact that a real world camera's motion is most likely unconstrained especially when mounted on a mobile robot conducting SLAM. Considering only one algorithm was unable to estimate the epipolar geometry for this motion, for only one of the two images, we find support for the first hypothesis.

In general, there were problems in the baseline of performance for the algorithms. What should have been a perfect estimation of the epipolar geometry for the point cube was often riddled with errors, specifically for translational motion. These errors; however, may have occurred due to the coplanarity of points in the cube (i.e., the four point of any side are coplanar). This most likely caused degeneracies in the calculation of the epipolar geometry and led to higher errors. Hence, the point cube's results, although correctly calculated, may not be as useful when determining how camera motion effects the error in the epipolar geometry.

In contrast to the point cube, the point cloud allowed the algorithms to establish a much better baseline. For Ma's and H&Z's algorithm, all errors in the epipolar geometry were near zero. Unfortunately, not all algorithms returned reasonable estimates. In particular, throughout our experiments, the Structure and Motion toolkit did not return reasonable numbers and although included in the appendix, the results have been, and will be ignored. As mentioned in Chapter III, we suspect the problems in this algorithm come from hard-coded implementation bugs.

Turning attention to the cube's baseline for Intel's OpenCV `cvFindFundamentalMat` function call using the 8-Point algorithm, we find that a suitable estimate of the epipolar geometry was only made when the camera motion was along its optical axis (referred to as scaling in the tables). The baseline for translational and rotational camera motions was undefined due to a constraint enforced in the programming that prevented the return of a fundamental matrix when the values in the diagonal of the singular value matrix are less than `FLT_EPSILON` (1.192092896e-07F). In other words, the implementation prevents the calculation of a fundamental matrix if any of the singular values of the results of an SVD are zero, which occurred with these examples. Furthermore, the baseline for the unconstrained camera motion on the cube had an average L1 error of 0.3056 ± 0.7046 and a maximum error of 2.382. These results make this the worst performance of all algorithms tested in establishing a baseline for the cube (excluding the results from the SAM toolkit).

The similarities in the results for the different methods found in OpenCV can be explained. OpenCV provides two linear methods of determining the fundamental matrix: 7-Point and 8-Point algorithms. The other, non-linear methods available in OpenCV are robust methods, such as RANSAC. In fact, RANSAC's results for the cube were exactly that of OpenCV's 8-Point algorithm. When looking into the implementation we find that the RANSAC loop performs a 7-Point estimation of the fundamental matrix for a subset of correspondences and a quality check exactly the same as our measure for determining the error in the epipolar geometry. Once a subset of points and its associated estimate of the fundamental matrix are accepted as suitable for estimating the entire dataset, the algorithm marks all points as either inliers or outliers. The inliers are then used as input to OpenCV's 8-Point algorithm to calculate a final estimate of the fundamental matrix. With the limited combination of points for the cube, it is only reasonable to assume that all 8 points are inliers which would mean the answers should be identical.

Lastly, the performance of OpenCV may be directly linked to the quality of the Singular Value Decomposition (SVD) calculation. In our initial trials we found that Matlab's performance was subjectively better than any implementation in C. Upon further investigation we found that all implementations in C deviated from Matlab's results when the SVD was calculated. The difference in the calculation was isolated to the last two columns of the right orthogonal matrix. All other columns were calculated identically. To overcome this problem we attempted to bypass the SVD in OpenCV and use the SVD from LAPack, CLApack as well as a compiled version of Matlab's SVD. All failed to recreate the same SVD calculations oibtained in Matlab proper. When hardcoding the results of the SVD from Matlab into our OpenCV implementation we were able to recreate the same results in C as its Matlab counterpart.

In summary, there appears to be enough support that rotation and scaling motions allow for better errors in estimates of the epipolar geometry, than translational motion. And that in fact, translational motion in the *xy*-plane increase the overall errors in the epipolar geometry. However, for a more conclusive study, the point cube should be replaced with another set of points, and possibly use more synthetic scene with varying number of points but similar in engineering to the point cloud.

VII. EXPERIMENT 3: ERROR IN THE INITIAL ESTIMATE OF THE FUNDAMENTAL MATRIX MEASURED BY REPROJECTION

A. DESCRIPTION

This experiment again focuses on determining the effect of camera motion on the errors in the epipolar geometry except at a point later in the projective reconstruction as described in Chapter III. Specifically, the error is calculated after a projective reconstruction is found. The measure of error is different for this experiment. We look instead at the Euclidean distance between the Cartesian coordinates of the original point correspondences to that of a reprojection using the projective reconstruction with the camera matrices *P* and *P'*. Hence, once *F* is initially calculated from corresponding pairs, the epipoles (e_1, e_2, or rather *e*, *e*') as well as the camera matrices can be derived. Having found *P* and *P'*, which encompass both the intrinsic and extrinsic parameters of the camera, we can find the camera centers (*C* and *C'*), which a ray can be found from each of the camera's center through the respective corresponding point (*x* and *x'*). These rays can then be extended out in front of the cameras in order to triangulate the location of the original 3-D points. These points are in a projective geometry (as opposed to a Euclidean geometry). The difficulty encountered in this triangulation is that the fundamental matrix is merely an estimation of the epipolar geometry. Therefore, when the rays are extended out into space, they do not tend to intersect as seen in Figure 20. So, the initial errors in the estimation of the fundamental matrix are compounded when it is used to estimate the epipoles (*e, e'*), the camera matrices (*P*, *P'*), and camera centers (*C, C'*), and further exasperated by the fact that the coordinates of the initial point correspondences contain noise as well.

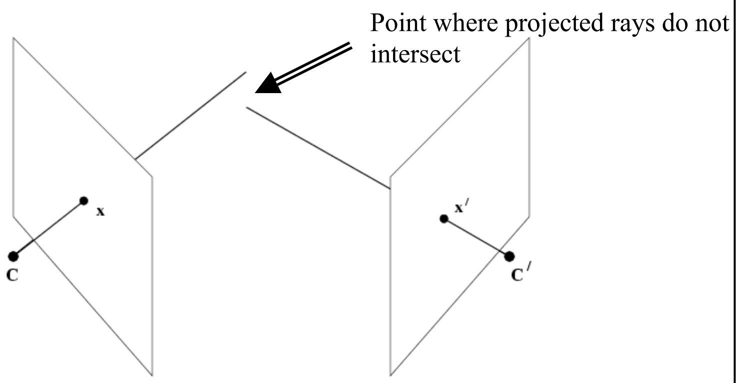


Figure 20. Error in Triangulation.

Depiction of the error that exists in triangulation using corresponding points (*x* and *x'*) and the centers of the cameras (*C* and *C'*). From: [1]

In order to compensate for the errors, one can use a more robust method of triangulation that does not require the rays to intersect, and will determine the point closest to both rays. The triangulated points are merely first estimates of the location of the original 3-D points in a projective space and can, in turn, be reprojected back onto the image planes by the same camera matrices (P, P') used for triangulation which were derived from the initial estimate of F. The Euclidean distance between the reprojected $(\tilde{x}, \tilde{y})$ and the original (x, y) is the measure of reprojection error for this experiment. The justification for this method of determining error is that the triangulated points are in a projective space, so the concept of closest is only defined up to the invariant properties of projective space versus a Euclidean space. Once the points are reprojected then the total error as defined above reflects on the quality of the estimated triangulation, which reflects on the quality of the estimated camera matrices, which reflects on the quality of the camera centers and epipoles, which lastly reflects on the quality of the initial estimate of the fundamental matrix.

The importance of the reprojection error is not only to measure the quality of the fundamental matrix, but it can also be used as a cost function in robust algorithm. For instance, the 3D reconstructed points can be classified as inliers and outliers based upon the error in their reprojection. The inliers can then be used to refine the initial estimate of F.

B. METHOD

Utilizing the same pairs of projections described in experiment two, as well as the same algorithms, we estimated the fundamental matrix (*F*) to produce a three-dimensional projective reconstruction using the projective reconstruction algorithm found described in Chapter III. Once these points are found we can the reproject them back onto the image plane using the estimated camera matrices *P* and *P'* such that

$$\vec{x'} = P'X$$
$$\vec{x} = PX$$

where X is the estimated 3D point in projective space. We then calculated the error in the reprojection by measuring the Euclidean distance between the original points and reprojected points as

$$\sum_{1}^{n} \sqrt{\left(x_{orig} - x_{proj}\right)^2 + \left(y_{orig} - y_{proj}\right)^2}$$

where n is the number of point correspondences, (x_{orig}, y_{orig}) is the original point in the image and (x_{proj}, y_{proj}) is the reprojected point into the same image using the estimated camera matrices.

C. RESULTS

A complete list of results can be found in Appendix B (Tables 53-108) along with a primer describing the format and content of the tables. The format is similar to that of experiment 2, where the main difference is that we have two sets of data for each set of point correspondences. That is, because we are reprojecting back into the image plane using *P* and *P'*, we are taking the Euclidean distance for points in both image one and image two. Hence, to capture this data we separate these experiments into two different tables so each set of noisy and noiseless projections of synthetic scenes results in two tables. For example, Table 4 and 5 are the results from a single projective reconstruction showing the errors in image one and image two respectively.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.076	0.0107	0.0738	0.1850
Pure Rotation	0.0179	0.0141	0.0778	0.3224
Pure Scaling	0.0099	0.0094	0.0707	0.1930
Unconstrained	0.0064	0.0070	0.0410	0.1342

Table 4. Image 1 reprojection error of the Cloud: Noisy Points dataset using Hartley and Zisserman method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	4.7646×10^{-07}	6.5768×10^{-07}	4.4761×10^{-06}	$1.1467 \text{ X } 10^{-05}$
Pure Rotation	1.5188×10^{-05}	1.1999×10^{-05}	6.6200×10^{-05}	0.0003
Pure Scaling	0.0002	0.0002	0.0016	0.0042
Unconstrained	3.6642×10^{-06}	4.042×10^{-06}	2.4164×10^{-05}	7.7048×10^{-05}

Table 5. Image 2 reprojection error of the Cloud: Noisy Points dataset using Hartley and Zisserman method to estimate F.

Overall, the minimum error found was zero[6] which was calculated multiple times for multiple camera motions (as seen in Table 5). The maximum error was 36.9641 calculated for Unconstrained motion between projections of the noisy point cube using Ma’s algorithm. (Table 95) The associated distribution of L1 errors was 5.4394±12.8772. This was also the maximum error for all Unconstrained motions. As far as the category Pure Translation, the minimum error calculated was zero to a maximum of 18.3802 for projection of the noisy point cube by H&Z.(Table 72) Pure Rotation and Pure Scaling again had minimums of zero, but maximums of 0.5317 and 0.4511 both from the

[6] Here zero is inclusive of values close to zero.

projection of the noisy point cube under Ma's algorithm. Hence, Ma's algorithm had three of the four maximum errors in the experiment all calculated using the point correspondences from the noisy point cube.

With regards to the point cube itself and establishing a baseline of performance for the algorithms, H&Z's results were equivalent to the ground truth in both images for Pure Rotation, and Pure Scaling. However, for Unconstrained and Pure Translation motions, the errors for image one were 0.6135±0.4558 and 1.0050±0.9367 respectively (Table 60), and for image two, 0.7366±0.9854 and 0.0902±0.0984 respectively.(Table 61)

Along the same lines, Ma predicted the ground truth for all camera motions except Pure Translation where its errors were 0.0220±0.0317 for image one, and 1.11498 ±1.4815 for image two.(Table 92 and 93) As with experiment two, OpenCV reported the same resulting errors for its robust and linear estimators, and was unable to estimate the fundamental matrix for when the motion was Pure Translation and Pure Rotation.(Tables 68-69, 76-77, and 84-85) It was; however, able to predict the ground truth for Pure Scaling motion but under Unconstrained motion, the results were mixed. For image one, the errors were 0.0741±0.0838 with a maximum of 0.2126.(Table 69) Conversely, for image two the errors were 0.0006±0.0013 with a maximum of 0.0039 which could be considered zero.(Table 69)

In comparison, the errors derived from the noisy point cube, H&Z had errors ranging from zero to a maximum of 18.3802. (Table 62 and 63) Specifically, for both image one and two, it was able to accurately estimate the epipolar geometry for the camera motions Pure Rotation, Pure Scaling and Unconstrained. However, the errors in the epipolar geometry of image one under Pure Translation motion was 1.1988±0.9519, and for image two, 3.0112±6.2841.(Table 62 and 63)

Ma's results for projections of the noisy cube ranged from zero to 36.9641. (Table 94 and 95) For image one, the ground truth results for the epipolar geometry was calculated for Pure Rotation and Pure Scaling motions; however, Pure Translation and Unconstrained motions had errors of 0.0722±0.0953 (maximum 0.2175) and

0.1634±0.2861 (maximum 0.7863). For image two, Pure Translation saw the best results of 0.0216±0.0314. Pure Rotation, Pure Scaling and Unconstrained motions were worse with a distribution of the errors ranging from 0.2135±0.1736(Pure Scaling) to 5.4394±12.8772(Unconstrained).(Table 95)

When looking at OpenCV's results based on the projection of the noisy point cube we find that there was deviation between the LMEDS approach and the RANSAC/8-Point methods (which scored the same). Although their results were different quantitatively, their actual ability to predict the epipolar geometry were nearly identical. For image one the epipolar geometry was predicted accurately for all three under Pure Rotation, and Pure Scaling motions; however, RANSAC and 8-Point also did so for Unconstrained camera motion, where LMEDS had errors of 0.0215±0.0608 (maximum 0.1721). (Tables 70, 78, and 86). For Pure Translation motion, none of OpenCV approaches used were able to predict the epipolar geometry with RANSAC and 8-Point returning 4.3826±4.9713 (maximum 16.0069)(Tables 70 and 78) and LMEDS errors of 1.2550±3.5488 (maximum 10.0379).(Table 86) Conversely, for the reprojection of the points into image two, the epipolar geometry was calculated correctly for every camera motion.(Tables 71, 79, and 87))

As far as establishing a baseline based on the projection of the noisy point cloud, we found that Ma's and H&Z's algorithms were able to predict the epipolar geometry for Pure Rotation, Pure Scaling and Unconstrained motions for both image one and two. Similarly, the baseline for the motion Pure Translation for image two was well established, but for image one it was 0.2769±0.5781 (maximum 3.6593) for H&Z, and 0.1029±0.2772 (maximum 2.7441) for Ma.(Tables 64 and 96) OpenCV, once again, returned the same results for all three algorithms, and was only able to predict the ground truth for Pure Scaling and Unconstrained motions in image one. When running the experiment using the motion Pure Rotation, the algorithms failed to return an estimate of the fundamental matrix, for both image one and two. The algorithms were able to predict the ground truth for Pure Translation when using the calculated *P*' (image two), but were not able to do the same for *P* (image one) which had an error distribution 0.6232±0.5740 (maximum 2.9356).(Tables 72-73, 78-79 and 88-89).

Both Ma's and H&Z's algorithms were able to predict the epipolar geometry from the projections of the noisy point cloud for image two, with a maximum error of 0.002. However, for image one, the errors were much greater. Ma's Unconstrained motion had a distribution of 0.3289±0.0991 (maximum 0.77).(Table 98) H&Z on the other hand had errors that can be interpreted as estimating the epipolar geometry where the greatest error was that of Pure Translation with a distribution of 0.076±0.0107 (maximum 0.0738).(Table 66)

D. DISCUSSION

With regard to the effect camera motion has on the errors in the epipolar geometry, we find that for the noisy point cube, the errors for pure translational motion results in the greatest errors for all algorithms (excluding SAM). The same is true for the projections of the noisy cloud; however, the errors are much less pronounced. For Ma, the greatest error was in the scaling motion for the second image, and rotational motion for the first image. Conversely, OpenCV had the greatest error in pure translational motion, yet H&Z based algorithm had equivalent errors for all three pure motions. So our results seem to tell us that for a small number of tracked features, the translational motion will result in the greatest errors. When any of the reviewed algorithms are tracking a large number of noisy points, the camera motion apparently has no effect on the resulting errors.

What we seems to be true from the results is that the more points that can be tracked the less likely camera motion has any effect on the errors. Hence, when any of the reviewed algorithms are tracking a large number of noisy points, the camera motion apparently has little effect on the resulting errors.

One improvement on our approach would be to use as a target of reprojection the points that are predicted by the fundamental matrix. Essentially, a similar error to what we used for experiment one and two which is a measure of how close it is to the epipolar line where the true point should lie. This is compared to the approach we used in which the error was derived by the distance between the original point in the image and the reprojected point.

THIS PAGE INTENTIONALLY LEFT BLANK

VIII. EXPERIMENT 4: NON-LINEAR OPTIMIZATION OF INITIAL ESTIMATION OF THE FUNDAMENTAL MATRIX

A. DESCRIPTION

An estimated fundamental matrix is nothing more than a model that explains the relationship between measured point correspondences and an epipolar geometry that exists between images. Calculating the fundamental matrix using the Direct Linear Transformation (DLT) algorithm as is done in the 8-Point algorithm can prove to be very efficient but, as results have shown in the last few experiments, errors persist from noise that perturbs the original correspondences. The literature suggests using the estimate of the fundamental matrix and the projective reconstruction merely as an initial estimate from which to start a total least squares regression. [1]

This experiment tests the quality of the epipolar geometry for each camera motion after a nonlinear regression algorithm, Levenberg-Marquardt (LM), is applied for each camera motion using differing cost functions. The details of the LM algorithm are in Chapter II. In short, it is a cross between Gradient descent and Gauss-Newton gradient descent.

There are three important inputs to the iterative regression process: parameterization of the inputs, the cost function that defines the quality of the current estimate, and the initial estimate (i.e., starting point in the hyperplane). Once these have been defined, the LM algorithm is used to refine the initial estimate stepwise until a threshold is met based on the error returned by the cost function. [1]

There are two places in which this method of optimization can be used in the projective reconstruction algorithm, each with different cost functions. The first is immediately after the initial estimate of the fundamental matrix is calculated. Using optimization at this time requires the use of the Sampson Distance. The second is after the projective reconstruction is calculated where we use the Gold Standard algorithm[1], to optimize the errors over the reprojected points described below.

B. METHOD

1. Optimization Using Sampson Distance.

As stated above, the three important parts of using the LM algorithm are the parameterization, cost function and initial estimate. The Sampson distance[1] is the cost function for this method of optimization and is given by

$$\sum_i \frac{\left(x_i'^T F x_i\right)^2}{\left(Fx_i\right)_1^2 + \left(Fx_i\right)_2^2 + \left(F^T x_i'\right)_1^2 + \left(F^T x_i'\right)_2^2}$$

where F is the fundamental matrix, and **x**, and **x'** are the 2D point correspondences (***x*⇔*x'***) as homogenous vectors and the distance is calculated for each point correspondence and the total is summed and represents the overall error in the epipolar geometry for the estimated *F*. This relates to the error as discussed in Chapter II, which was used in both experiment one and two in that it leverages the same concept of the relationship between a point and its associated epipolar line.

The parameterization of the Fundamental matrix for this experiment is

$$F = \begin{bmatrix} a & b & \alpha a + \beta b \\ c & d & \alpha c + \beta d \\ \alpha' a + \beta' c & \alpha' b + \beta' d & \alpha' \alpha a + \alpha' \beta b + \beta' \alpha c + \beta' \beta d \end{bmatrix}.[1]$$

The intitial estimate of the fundamental matrix comes from the 8-Point algorithm as described by [1].

2. Optimization Using Gold Standard Method

For the Gold Standard optimization, the input is the estimation of the second camera (*P'*), the 4x1 homogenous representation of the 3D points from triangulating into projective space, and the original point correspondences. The LM algorithm iteratively manipulates both *P'* and the 3D points and the original image coordinates are used as the target of optimization. The output of this optimization is a new *F*. [1]

The cost function for this method is based upon a measure of error in the reprojection of the projective reconstruction using *P* (identity matrix augmented with a zero vector) and *P'* (calculated as part of the projective reconstruction). We define the reprojection error as the Euclidean distance between the original Cartesian coordinate of the feature in the respective image and the reprojection of the reconstructed popints in projective space.[1] Hence for each iteration of the LM algorithm, the entries in *P'* and X_i for i=1 to *n* (where *n* is the number of triangulated 3D points) are adjusted and then used to calculate an $x_{est,i}$ such that $x_{est,i}=P'X_i$. The summation of the Euclidean distance between the original point in the image (x) and the reprojected point (x_{est}) is the quality of the current estimate of *P'* and X_i for all *i*. This process is repeated for the other image using projection matrix *P*. However, as discussed below, the cost function does not calculate the summation due to our method of implementing this algorithm.

Although we are iteratively adjusting both the camera matrix and the projective 3D points, we actually are able to output a new estimate of *F*. Once the LM algorithm exits based on either reaching a minimum acceptable error, or a maximum number of iterations, we have the final estimate of *P'* and for all *i*, X_i. We extract the last column of the camera matrix *P'*, i.e., $\vec{t}$ which is the camera center in world coordinates, and transform it into its equivalent skew-symmetric form. We then multiply it by first three rows of *P'* to find the new *F*. Specifically,

$$P =: \left[R \,|\, \vec{t} \right] \text{ and } \vec{t} =: \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

$$\text{then } [\vec{t}]_x = \begin{bmatrix} 0 & -c & b \\ c & 0 & -a \\ -b & a & 0 \end{bmatrix}$$

$$F_{new} = [\vec{t}]_x \times R$$

After: [1].

3. Implementation

In order to implement the algorithms as described above, we utilize Matlab's `LSQNONLIN` function. It is a nonlinear solution solver which can be called with the Levenberg-Marquardt option turned on. The parameterized inputs are passed into the method as the initial estimates along with a pointer to the cost function, defined in a separate .m file. One thing to note about this form of implementation is that the `LSQNONLIN` implementation seeks to minimize the sum of the squared errors of whatever is returned by the cost function. Hence, we return a matrix where each entry is derived from an individual correspondence. For example, when utilizing the Sampson Distance as the cost function, the returned matrix by the cost function script is made up of each individual quotient of the function shown in section one above. That is,

$$\frac{\left(x_i^{'T} F x_i\right)^2}{\left(Fx_i\right)_1^2 + \left(Fx_i\right)_2^2 + \left(F^T x_i^{'}\right)_1^2 + \left(F^T x_i^{'}\right)_2^2} = error_i$$

and $error_i$ is a single entry in the returned matrix.

In general the projective reconstruction algorithm used in this experiment parallels the algorithm presented in Chapter III. However, this time we seek to optimize with respect to the fundamental matrix when possible. Specifically, we determine how the camera motion plays into the optimization by not performing optimization concurrently. That is, we first perform a full projective reconstruction starting from scratch and only optimize after the initial estimation of the fundamental matrix using the Sampson distance. We then perform another full projective reconstruction and this time only optimize after the projective reconstruction is complete using the Gold Standard method.[7] For both iterations through the complete algorithm we use the same corresponding points from the different camera motions. This breakdown effectively allows us to determine the effect that camera motion has on each form of optimization.

[7] For more details see the flow chart for the projective reconstruction in Chapter III, section B.

C. RESULTS

Again, we utilize the same general form of the table to present the results of this experiment, calculating L1, L2 and max statistics from the error data and each set of statistics is tabled by virtual scene in Appendix IX (Tables 98-105). The most obvious difference in the way the results are reported between what was done in last experiment and this experiment is that we have added a row that has the pre-optimization values for the respective cost function and camera motion for ease of comparison. As an example, Table 6 is shown below.

Motion	Average	Standard Deviation	Max	L2 Norm
Translation: Before				
Translation: After				
Rotation: Before				
Rotation: After				
Scale: Before				
Scale: After				
Unconstrained: Before				
Unconstrained: After				

Table 6. Sample Table.

Here the Average and Standard Deviation relate to L1 statistics, max is the max of the absolutes and L2 is calculated from the original error data. Our interest is primarily in the camera motion between projections of the noisy cloud due to its relationship with scenes the real world.

With regard to the unit of measurement of the errors, both methods of optimization result in a new estimate of the fundamental matrix, hence we utilize the measure of error defined by the equation $x'^{T}Fx=error$.

Motion	Average	Standard Deviation	Max	L2 Norm
Translation: Before	0.0087	0.0122	0.0894	0.2122
Translation: After	0.0104	0.0116	0.0660	0.2192
Rotation: Before	0.0233	0.0181	0.0814	0.4160
Rotation: After	0.0096	0.0083	0.0382	0.1782
Scale: Before	0.0222	0.0212	0.1242	0.4326
Scale: After	0.0190	0.0278	0.2045	0.4744
Unconstrained: Before	0.0109	0.0132	0.0749	0.2415
Unconstrained: After	0.0378	0.0375	0.2069	0.7503

Table 7. Sampson Distance Optimization: Noisy Cloud Projections.

Comparison of the error in the epipolar geometry before and after the minimization of the Sampson Distance for the data set Cloud: Noisy Points

1. Optimization Using Sampson Distance.

As far as a baseline of performance, optimization for both virtual scenes was able to find the ground truth for the epipolar geometry for all camera motions with the most error under Pure Translation motion, which had had a maximum error of 0.0101

What is most interesting is the results from projections of the noisy cloud due to its relationship to the real world. For Table 7, we can see that for rotational motion, both the average and standard deviation in the L1 error decreased after optimization. Likewise, the Max and L2 Norm of the error decreased for the same motion. However, for translational motion, the max increased, the average L1 stayed constant, and the deviation in the L1 error decreased. For the same motion the L2 error decreased. For scaling, the average error decreased, but the standard deviation, as well as the maximum error increased. Unconstrained motion saw all error increase across the board,

2. Optimization Using Gold Standard Method

For the most part our baseline is well defined except with regards to translational movement. Under pure translation we find that the use of the Gold Standard method degraded the fit of the model considerably. For the noiseless cube projections, the Euclidean distance went from an average of 0.0890 prior to optimization, to 66.7260 after optimization with a max of 226.8403. Likewise, with regard to translational camera motion, the optimization on the noiseless cloud projections changed the average pre-optimization of $4.3668\text{x}10^{-16}$ to $1.2150\text{x}10^{+15}$ post-optimization with a greater max and L2 values as well. If all camera motions experienced the same dramatic effect, we would consider that it was an error in our implementation, but seeing as the other values are reasonable the only explanation is that this form of optimization with this cost function should not be performed under pure translational motion. That said, it would be interesting it perform this experiment as a robust estimation method and classifying points as outliers.

Regardless of the obvious issues with translational motion in our noiseless projections, the errors found for the other camera motions are what is expected and specifically, the errors for the unconstrained motion decreased. That is, the average L1 error for the projections of the noiseless cube under unconstrained motion was 0.7366 pre-optimization and $7.6233\text{x}10^{-08}$ post optimization.

D. DISCUSSION.

1. Optimization Using Sampson Distance

Again we see in our results that the algorithms have problems with translation motion when we attempt to define our baseline. What we are most interested in is how the statistics of our results changed for the errors from the projections of the noisy cloud.

For the noisy point cloud under rotational motion, the results were improved across the board but the converse is true for unconstrained motions. For translation and scaling, which are related by the fact that they both create parallax between the images, their results are inversely related. Where translation saw an increase in average error, scaling saw a decrease and the reverse is true for their standard deviations and maximum error.

Since scaling is this the expected motion of a robot with a camera that is facing forward, we find that we do not have to do optimization under forward (scaling) motion. That is, we would not want to perform optimization after the initial estimate of the fundamental matrix. Along the same lines, legged robots such as the Sony Aibo simply cannot perform pure scaling motion even on the flattest surface. The bobbing of the head could provide enough translational and rotational movement that optimization may still be warranted, but as can be seen, the optimization on the unconstrained motion dramatically degraded the fit of the epipolar geometry. For better understanding, testing on real robots is warranted.

Furthermore, we find that after optimization with respect to the Sampson distance, when attempting to label reconstructed points as outliers after the optimization process, the points with errors near the max could be classified as outliers which may improve the projective reconstruction by their exclusion.

2. Optimization Using Gold Standard Method

As far as a baseline, we were unable to establish a baseline near the ground truth for translational motion which results in outlandish errors after optimization. This is not related to the coplanarity of the point in the point cube as before, because every projection of the synthetic scenes results in the same unrealistic increase in errors. All other results of our baseline were reasonable.

What we can take away from this is that when we are tracking a large number of points, we do not want to perform optimization assuming a robot's motion is inherently unconstrained. But, as the number of points decreases, and the features are tracked better (relatively noiseless) we find an improvement in the error across the epipolar geometry, so we can use the Gold Standard method but the cost of doing so, and the benefit is perhaps not worth the effort.

IX. EXPERIMENT 5: PROJECTIVE TO EUCLIDEAN UPGRADE (CALIBRATED)

A. DESCRIPTION

After producing a projective reconstruction of the original scene as measured by our previous experiments we can, from the right perspective, i.e., the original cameras' position, view the reconstruction and see the original images. However, any change in perspective may leave the view of the scene with little resemblance to the original Euclidean scene. In order to move from a projective to Euclidean geometry the reconstruction algorithm needs to acquire more information. This can be from an external source that provides the calibration information of the camera, or from the scene itself. For this experiment we focus on upgrading the geometry to Euclidean by having the information provided by the calibration matrix *K,* described in Chapter II. However, we take it one step further in that we will utilize the resulting Euclidean information to determine the translation of an Aibo robot from the world origin.

The calibration information of the camera (*K*) can upgrade the fundamental matrix to a more specific essential matrix (*E*) where $E = K'^{T} FK$. Here *E* is more specific than *F* since it is unique for each *K*. Likewise the projection matrix *P*, $P = K\left[R \mid \vec{t}\right]$, can be upgraded to a partially calibrated camera matrix, $\hat{P} = K^{-1}P = \left[R \mid \vec{t}\right]$. This partially calibrated camera is referred to as the pinhole form of the camera and removes any effect that the camera's intrinsic parameters had on the original corresponding points. This capability is specific to a calibrated camera scenario since the calibration matrix *K* is known a priori. The residual augmented matrix composed of *R* and ***t*** describe the rotation and translation of the camera plane from the world origin and are called the camera's extrinsic parameters. However, obtaining the camera's center is more complicated then simply declaring the vector ***t*** to be the center of the second camera. [1]

The camera location in space is defined by the entries in the vector C=(*X,Y,Z,*T)^T. Here, *X,Y* and *Z* are the respective location in the world reference frame and the value of T is the homogenous value for the point. The entries are calculated by finding the determinant of the *P* matrix. For example, if $P' = \left[\overline{p_1}, \overline{p_2}, \overline{p_3}, \overline{p_4}\right]$, then the *X* value for the camera center is $X = \det\left(\left[\overline{p_2}, \overline{p_3}, \overline{p_4}\right]\right) / T$, and $T = -\det([p_1, p_2, p_3])$. [1]

B. METHOD

In order to recover the extrinsic parameters of the second camera with respect to the first, we need to find *P'*, which is done by performing a projective reconstruction on the features tracked between two cameras' images of the same scene. For this experiment, we utilized the capabilities of the Sony Aibo dog (robot). By performing calibration on its nose camera, we found the intrinsic camera calibration matrix *K* (shown below in the form discussed in Chapter II). [30]

$$K = \begin{bmatrix} \alpha_x & & p_x \\ & \alpha_y & p_y \\ & & 1 \end{bmatrix} \rightarrow K = \begin{bmatrix} 198.803 & & 107.3896 \\ & 196.871 & 85.0953 \\ & & 1 \end{bmatrix}$$

Knowing the calibration matrix, we then captured two images from the Aibo's nose camera. The ground truth with respects to the camera's positions (localization) for the two images is a pure translational shift in the camera plane (along the x-axis) of 13.5 inches. We accomplished this by positioning the Aibo for each image such that it was in the sitting position with all servos/motors on. This locks its appendages and limits undesired movement while handled between images. Once in the desired position, we capture an image with the Aibo's nose camera. Although we are striving for pure translational movement, undesired motion (noise in its motion) is unavoidable. There could easily be a slight, unnoticeable change in the pose of the sitting Aibo when handling it. For instance there could be a rotation from a slight turn of its neck, or perhaps the joints in the legs move when picking it up or placing it down. Whatever the case, the ground truth is only as accurate and precise as handling any robot. Therefore, the approximate movement was pure translational along the x-axis, but if the results have any slight change in either the y- or z-directions, we cannot account for these subtleties.

Figure 21 shows the original jpeg images from the Aibo, as well as images with the tracked features. As in the first experiment, we tracked the points by utilizing the Lucas Kanade Optical Flow algorithm in OpenCV. Furthermore, the fundamental matrix describing the epipolar geometry between the scenes was calculated using OpenCV's 8-Point algorithm.

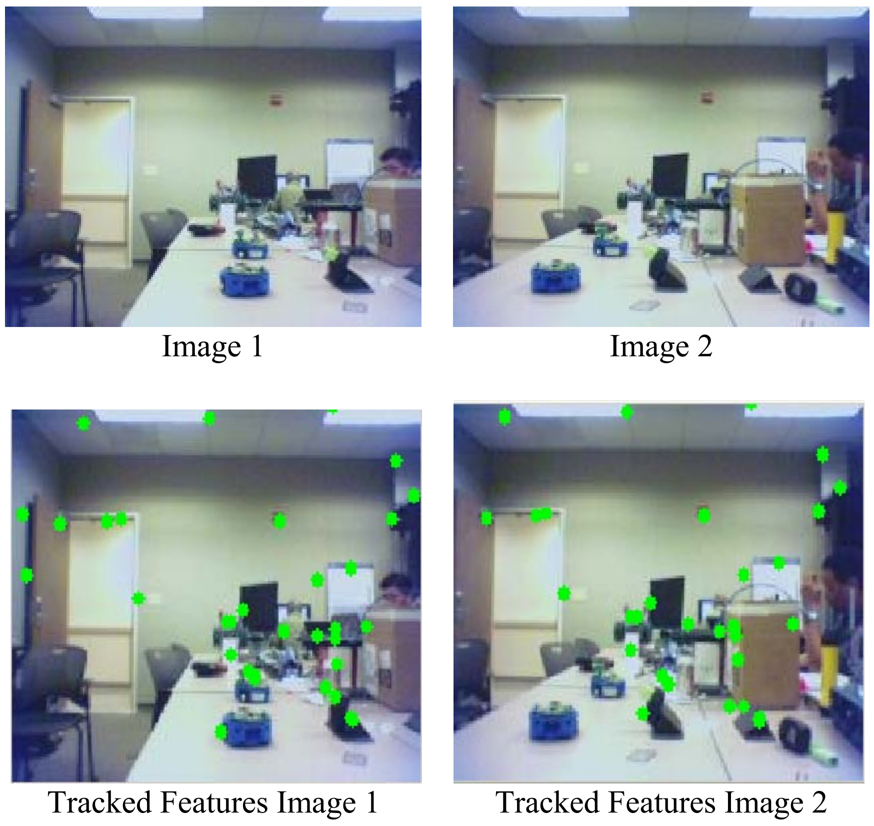


Figure 21. Tracked Features In Aibo Images.

Original Aibo images and points tracked between images using the multi-pass feature tracking. Motion between captures was ~13.5" translation with ~0 degrees rotation.

C. RESULTS

Using the point correspondences tracked between images, we performed a projective reconstruction as described in Chapter III. Knowing the calibration matrix, we upgraded the estimated camera matrix *P*' to its pinhole equivalent such that $K^{-1}P'=[R|t]$. Both the camera matrices are:

$$P=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\end{bmatrix} \qquad P'=\begin{bmatrix}0.0074 & 0.0090 & 2.1653 & -0.5375\\ 0.0074 & -0.0174 & 1.7022 & -0.4310\\ -0.0144 & -0.0017 & -3.9394 & 1.0000\end{bmatrix}$$

where *P* is the initial camera matrix, and *P'* is the partially calibrated estimate of the second camera.

The vector ***Center*** for the first camera was assumed to be at the world origin ($[0,0,0]^T$). The vector $\boldsymbol{Center_{P'}}$ of the second camera was calculated as follows.

$$\begin{aligned}
&P'=\left[\overline{p_1},\overline{p_2},\overline{p_3},\overline{p_4}\right]\\
&X=\det\left(\left[\overline{p_2},\overline{p_3},\overline{p_4}\right]\right)\\
&Y=-\det\left(\left[\overline{p_1},\overline{p_3},\overline{p_4}\right]\right)\\
&Z=\det\left(\left[\overline{p_1},\overline{p_2},\overline{p_4}\right]\right)\\
&T=-\det\left(\left[\overline{p_1},\overline{p_2},\overline{p_3}\right]\right)\\
&Center_{P'}=[X/T,Y/T,Z/T]^T\\
&Center_{P'}=\left[1.5461,0.1031,-0.0057\right]^T\times10^{16}
\end{aligned}$$

[1]

D. DISCUSSION

Figure 22 shows a plot of the two camera centers as seen in the *xy*-plane. Knowing the ground truth of the Aibo's position for both images, we expected a result that had a large value in the *x* direction with respect to both the *y* and *z* values. The second camera's estimated center is ($[1.5461, 0.1031, -0.0057]^T\times10^{16}$), which shows a translation in the (*x,y,z*) direction from the origin $(0,0,0)^T$. The x10^{16}, although alarming in its magnitude, can be assumed to be one. This is because the general scale is unknown.

Looking at the *x, y* and *z* values, the value 1.5461 is, in fact, relatively large when compared to 0.1031 and -0.0057 for the y and z directional translations respectively, hence at first glance it would seem that we have estimated the Euclidean movement of the Aibo. Scaling the 1.5461 to represent 13.5 inches, the value 0.1031 becomes 0.9 inches and -0.0057 about -0.05 inches. The question then is if the change in *y* and *z* directions makes sense. For the *z* direction, -0.05 inches is believable because it is small enough that we can assume that we were especially careful when lining up the Aibo's nose between images with respect to the z-axis. As for the *y* component of $Center_{P'}$, the movement of 0.9 inches is a little more difficult to explain. However, it is reasonable to believe that the process of lifting the Aibo and setting it down caused the joints in the front legs, although its servos were locked, to move such that there was less than an inch movement with respect to the *y*-direction. Hence, we can safely assume that for these images we were able to acquire some idea of how the Aibo moved, although this is only true in a Euclidean sense since the scale is still unknown.

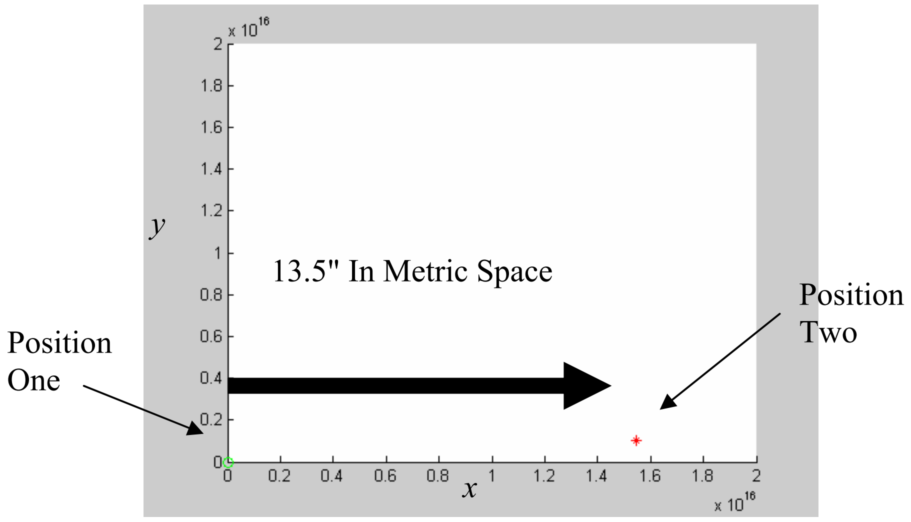


Figure 22. Euclidian Localization.

Above is a two-dimensional Cartesian coordinate space showing the translation of the camera (red '*') from the origin (green 'o'). It is a Euclidean localization since the metric measurement of 13.5" does not correspond to the 1.55 x 10^{16} unit of measurement of the x axis.

X. EXPERIMENT 6: PROJECTIVE TO EUCLIDEAN UPGRADE (UNCALIBRATED)

A. DESCRIPTION

Once a fundamental matrix is established and (optionally) iteratively improved, three-dimensional points triangulated, the resulting 3D structure is in a projective space. Again, to upgrade this to a Euclidean or metric geometry we need additional information. For the previous experiment, this came in the form of the calibration matrix *K*. However, for this experiment we are dealing with uncalibrated cameras so we can not simply upgrade the camera matrix to a pinhole model [1].

In order to acquire additional information we have two choices: we can analyze the scene and retrieve points and planes at infinity, or have something prepositioned in the scene to cue from. The latter method uses ground truth to upgrade the geometry of the reconstruction from projective points (X_P) to their Euclidean equivalent (X_E). There are two possibilities as recommended by H&Z. The first involves finding a map (H_{map}) between known ground truth points and their 3D projective equivalents. Then for each point (*i*), we can use the map such that, $X_{Ei}=H_{\mathrm{map}}X_{Pi}$ for i=1, 2, 3, …, n. The second method for upgrading using ground truth is similar to the first, except this case uses the relation between the known Euclidean point, as it relates to its 2D equivalent in the image plane. Although we know ground truth locations of features, our goal here is to simply attain a reconstruction up to scale, i.e., Euclidean reconstruction.[1]

For the first method, knowing the ground truth alone is a must for this algorithm so this approach will not work for SLAM in an unknown environment. However, in a controlled environment there could be unique markers that are recognized in the image, with each corresponding to a single known position in Euclidean space. By including these markers as tracked features we can declare their image location as corresponding points and include them in the projective reconstruction. The reconstruction will eventually triangulate to find the 3D point in projective space which means we now have the ground truth points' projective coordinate and known Euclidean equivalent. Hence, with at least five of these geometric correspondences, we can find H_{map} using a Direct Linear Transformation (DLT). Once found, H_{map} can be used to upgrade all the projective points without a Euclidean counterpart assuming that no three ground truth points are coplanar.[1]

For the second method, in order to project the Euclidean point onto the image plane we need projection matrix P'. But when the calibration matrix K is unknown, we cannot account in the projection for the unknown noise caused by the lens. Knowing that the projection matrix is 3x4, there is an unknown 4x4 homography that need be applied to the homogenous Euclidean points (4x1 vectors) prior to multiplying by the estimated P'. It is this homography which can then be used in conjunction with P' to upgrade our projective reconstruction to its Euclidean equivalent. That is, $x = P' \times H^{T} \times X_E$ where P' is the calculated camera matrix. [1]

B. METHOD

1. Ground Truth Based Upgrade Using a Single Homography (map) ($\mathbf{X_{Ei}=H_{map}X_{Pi}}$ *for all* i)

Here, $X_{E,i}$ and $X_{P,i}$ are 3D homogenous points (4x1 vectors) in Euclidean and projective geometries respectively. The derivation of how to find H_{map}[1] begins as follows:

$$\boxed{\begin{array}{l} \forall_i \overrightarrow{X}_{E,i} = H\overrightarrow{X}_{P,i} \\ H = \begin{pmatrix} \overrightarrow{H}^{1,T} \\ \overrightarrow{H}^{2,T} \\ \overrightarrow{H}^{3,T} \\ \overrightarrow{H}^{4,T} \end{pmatrix} \quad \overrightarrow{X}_{E,i} = \begin{pmatrix} X_E \\ Y_E \\ Z_E \\ W_E \end{pmatrix} \quad \overrightarrow{X}_{P,i} = \begin{pmatrix} X_P \\ Y_P \\ Z_P \\ W_P \end{pmatrix} \end{array}}$$

where W_P and W_E are the homogenous values for the respective point. Likewise, the H matrix is redefined by its rows which are size 1x4. The remaining deviation is shown below.

$$0 = \vec{X}_{E,i} \times H \vec{X}_{P,i}$$

$$0 = \begin{pmatrix} X_E \\ Y_E \\ Z_E \\ W_E \end{pmatrix} \times \begin{pmatrix} \vec{H}^{1,T} \vec{X}_{P,i} \\ \vec{H}^{2,T} \vec{X}_{P,i} \\ \vec{H}^{3,T} \vec{X}_{P,i} \\ \vec{H}^{4,T} \vec{X}_{P,i} \end{pmatrix} \rightarrow \quad 0 = \begin{pmatrix} X_E \\ Y_E \\ Z_E \\ W_E \end{pmatrix} \times \begin{pmatrix} \vec{X}^{T}_{P,i} \vec{H}^{1} \\ \vec{X}^{T}_{P,i} \vec{H}^{2} \\ \vec{X}^{T}_{P,i} \vec{H}^{3} \\ \vec{X}^{T}_{P,i} \vec{H}^{4} \end{pmatrix} \rightarrow \quad 0 = \begin{pmatrix} X_E \\ Y_E \\ Z_E \\ W_E \end{pmatrix} \times \begin{pmatrix} \vec{X}^{T}_{P,i} \vec{H}^{1} \\ \vec{X}^{T}_{P,i} \vec{H}^{2} \\ \vec{X}^{T}_{P,i} \vec{H}^{3} \\ \vec{X}^{T}_{P,i} \vec{H}^{4} \end{pmatrix}$$

$$0 = \begin{pmatrix} \left(Y_E \vec{X}^{T}_{P,i} \vec{H}^{3} - \vec{X}^{T}_{P,i} \vec{H}^{2} Z_E\right) + \left(Y_E \vec{X}^{T}_{P,i} \vec{H}^{4} - \vec{X}^{T}_{P,i} \vec{H}^{2} W_E\right) + \left(Z_E \vec{X}^{T}_{P,i} \vec{H}^{4} - \vec{X}^{T}_{P,i} \vec{H}^{3} W_E\right) \\ \left(\vec{X}^{T}_{P,i} \vec{H}^{3} W_E - Z_E \vec{X}^{T}_{P,i} \vec{H}^{4}\right) + \left(Z_E \vec{X}^{T}_{P,i} \vec{H}^{1} - \vec{X}^{T}_{P,i} \vec{H}^{3} X_E\right) + \left(W_E \vec{X}^{T}_{P,i} \vec{H}^{1} - \vec{X}^{T}_{P,i} \vec{H}^{4} X_E\right) \\ \left(Y_E \vec{X}^{T}_{P,i} \vec{H}^{4} - \vec{X}^{T}_{P,i} \vec{H}^{2} W_E\right) + \left(\vec{X}^{T}_{P,i} \vec{H}^{2} X_E - Y_E \vec{X}^{T}_{P,i} \vec{H}^{1}\right) + \left(W_E \vec{X}^{T}_{P,i} \vec{H}^{1} - \vec{X}^{T}_{P,i} \vec{H}^{4} X_E\right) \\ \left(\vec{X}^{T}_{P,i} \vec{H}^{2} Z_E - Y_E \vec{X}^{T}_{P,i} \vec{H}^{3}\right) + \left(\vec{X}^{T}_{P,i} \vec{H}^{2} X_E - Y_E \vec{X}^{T}_{P,i} \vec{H}^{1}\right) + \left(\vec{X}^{T}_{P,i} \vec{H}^{3} X_E - Z_E \vec{X}^{T}_{P,i} \vec{H}^{1}\right) \end{pmatrix}$$

$$0 = \begin{pmatrix} Y_E \vec{X}^{T}_{P,i} \vec{H}^{3} - \vec{X}^{T}_{P,i} \vec{H}^{2} Z_E + Y_E \vec{X}^{T}_{P,i} \vec{H}^{4} - \vec{X}^{T}_{P,i} \vec{H}^{2} W_E + Z_E \vec{X}^{T}_{P,i} \vec{H}^{4} - \vec{X}^{T}_{P,i} \vec{H}^{3} W_E \\ \vec{X}^{T}_{P,i} \vec{H}^{3} W_E - Z_E \vec{X}^{T}_{P,i} \vec{H}^{4} + Z_E \vec{X}^{T}_{P,i} \vec{H}^{1} - \vec{X}^{T}_{P,i} \vec{H}^{3} X_E + W_E \vec{X}^{T}_{P,i} \vec{H}^{1} - \vec{X}^{T}_{P,i} \vec{H}^{4} X_E \\ Y_E \vec{X}^{T}_{P,i} \vec{H}^{4} - \vec{X}^{T}_{P,i} \vec{H}^{2} W_E + \vec{X}^{T}_{P,i} \vec{H}^{2} X_E - Y_E \vec{X}^{T}_{P,i} \vec{H}^{1} + W_E \vec{X}^{T}_{P,i} \vec{H}^{1} - \vec{X}^{T}_{P,i} \vec{H}^{4} X_E \\ \vec{X}^{T}_{P,i} \vec{H}^{2} Z_E - Y_E \vec{X}^{T}_{P,i} \vec{H}^{3} + \vec{X}^{T}_{P,i} \vec{H}^{2} X_E - Y_E \vec{X}^{T}_{P,i} \vec{H}^{1} + \vec{X}^{T}_{P,i} \vec{H}^{3} X_E - Z_E \vec{X}^{T}_{P,i} \vec{H}^{1} \end{pmatrix}$$

$$0 = \begin{pmatrix} -\vec{X}^{T}_{P,i} \vec{H}^{2} W_E - \vec{X}^{T}_{P,i} \vec{H}^{2} Z_E + Y_E \vec{X}^{T}_{P,i} \vec{H}^{3} - \vec{X}^{T}_{P,i} \vec{H}^{3} W_E + Y_E \vec{X}^{T}_{P,i} \vec{H}^{4} + Z_E \vec{X}^{T}_{P,i} \vec{H}^{4} \\ W_E \vec{X}^{T}_{P,i} \vec{H}^{1} + Z_E \vec{X}^{T}_{P,i} \vec{H}^{1} + \vec{X}^{T}_{P,i} \vec{H}^{3} W_E - \vec{X}^{T}_{P,i} \vec{H}^{3} X_E - Z_E \vec{X}^{T}_{P,i} \vec{H}^{4} - \vec{X}^{T}_{P,i} \vec{H}^{4} X_E \\ -Y_E \vec{X}^{T}_{P,i} \vec{H}^{1} + W_E \vec{X}^{T}_{P,i} \vec{H}^{1} - \vec{X}^{T}_{P,i} \vec{H}^{2} W_E + \vec{X}^{T}_{P,i} \vec{H}^{2} X_E + Y_E \vec{X}^{T}_{P,i} \vec{H}^{4} - \vec{X}^{T}_{P,i} \vec{H}^{4} X_E \\ -Z_E \vec{X}^{T}_{P,i} \vec{H}^{1} - Y_E \vec{X}^{T}_{P,i} \vec{H}^{1} + \vec{X}^{T}_{P,i} \vec{H}^{2} X_E + \vec{X}^{T}_{P,i} \vec{H}^{2} Z_E - Y_E \vec{X}^{T}_{P,i} \vec{H}^{3} + \vec{X}^{T}_{P,i} \vec{H}^{3} X_E \end{pmatrix}$$

$$0 = \begin{pmatrix} \vec{0}^{T} & -\vec{X}^{T}_{P,i} W_E - \vec{X}^{T}_{P,i} Z_E & Y_E \vec{X}^{T}_{P,i} - \vec{X}^{T}_{P,i} W_E & Y_E \vec{X}^{T}_{P,i} + Z_E \vec{X}^{T}_{P,i} \\ W_E \vec{X}^{T}_{P,i} + Z_E \vec{X}^{T}_{P,i} & \vec{0}^{T} & \vec{X}^{T}_{P,i} W_E - \vec{X}^{T}_{P,i} X_E & -Z_E \vec{X}^{T}_{P,i} - \vec{X}^{T}_{P,i} X_E \\ -Y_E \vec{X}^{T}_{P,i} + W_E \vec{X}^{T}_{P,i} & -\vec{X}^{T}_{P,i} W_E + \vec{X}^{T}_{P,i} X_E & \vec{0}^{T} & Y_E \vec{X}^{T}_{P,i} - \vec{X}^{T}_{P,i} X_E \\ -Z_E \vec{X}^{T}_{P,i} - Y_E \vec{X}^{T}_{P,i} & \vec{X}^{T}_{P,i} X_E + \vec{X}^{T}_{P,i} Z_E & -Y_E \vec{X}^{T}_{P,i} + \vec{X}^{T}_{P,i} X_E & \vec{0}^{T} \end{pmatrix} \begin{pmatrix} \vec{H}^{1} \\ \vec{H}^{2} \\ \vec{H}^{3} \\ \vec{H}^{4} \end{pmatrix}$$

$$0 = \begin{pmatrix} \vec{0}^{T} & -\left(W_E + Z_E\right) \vec{X}^{T}_{P,i} & \left(Y_E - W_E\right) \vec{X}^{T}_{P,i} & \left(Y_E + Z_E\right) \vec{X}^{T}_{P,i} \\ \left(W_E + Z_E\right) \vec{X}^{T}_{P,i} & \vec{0}^{T} & \left(W_E - X_E\right) \vec{X}^{T}_{P,i} & -\left(Z_E + X_E\right) \vec{X}^{T}_{P,i} \\ \left(W_E - Y_E\right) \vec{X}^{T}_{P,i} & \left(X_E - W_E\right) \vec{X}^{T}_{P,i} & \vec{0}^{T} & \left(Y_E - X_E\right) \vec{X}^{T}_{P,i} \\ -\left(Y_E + Z_E\right) \vec{X}^{T}_{P,i} & \left(Z_E + X_E\right) \vec{X}^{T}_{P,i} & \left(X_E - Y_E\right) \vec{X}^{T}_{P,i} & \vec{0}^{T} \end{pmatrix} \begin{pmatrix} \vec{H}^{1} \\ \vec{H}^{2} \\ \vec{H}^{3} \\ \vec{H}^{4} \end{pmatrix} \qquad [31]$$

The resulting coefficient matrix,

$$\begin{pmatrix} \vec{0}^T & -(W_E+Z_E)\vec{X}_{P,i}^T & (Y_E-W_E)\vec{X}_{P,i}^T & (Y_E+Z_E)\vec{X}_{P,i}^T \\ (W_E+Z_E)\vec{X}_{P,i}^T & \vec{0}^T & (W_E-X_E)\vec{X}_{P,i}^T & -(Z_E+X_E)\vec{X}_{P,i}^T \\ (W_E-Y_E)\vec{X}_{P,i}^T & (X_E-W_E)\vec{X}_{P,i}^T & \vec{0}^T & (Y_E-X_E)\vec{X}_{P,i}^T \\ -(Y_E+Z_E)\vec{X}_{P,i}^T & (Z_E+X_E)\vec{X}_{P,i}^T & (X_E-Y_E)\vec{X}_{P,i}^T & \vec{0}^T \end{pmatrix} \qquad \boldsymbol{H}=\begin{pmatrix} \vec{H}^1 \\ \vec{H}^2 \\ \vec{H}^3 \\ \vec{H}^4 \end{pmatrix}$$

is 4x16 and the unknown ***H*** vector is 16x1. We repeat this process for each correspondence between geometric space, and each coefficient matrix found becomes four rows in an *A* matrix. We can then find the non-trivial null space of the *A* matrix, using the fact that 0=*A**H***, which is the vector ***H***, and likewise can be reshaped into the matrix H_{map} by using row major ordering.

We perform a Direct Linear Transformation to calculate the vector ***H*** which is the 4x4 H_{map} matrix in row major order.

2. Ground Truth Based Upgrade Using a Homography Between the Known Euclidean Point and Its Projection by *P'*

Using $\vec{x}_i = PH^{-1}\vec{X}_i$ we can derive a 4x4 homography matrix H^{-1} which is the unknown in the relationship between the projected point $\vec{x}_i$, the original 3D point, $\vec{X}_i$, and the camera matrix *P* such that,

$$\vec{x}_i = \begin{pmatrix} x \\ y \\ w \end{pmatrix} \qquad \vec{X}_i = \begin{pmatrix} X \\ Y \\ Z \\ W \end{pmatrix} \qquad H^{-1} = \begin{pmatrix} \vec{H}_r^{-1,1,T} \\ \vec{H}_r^{-1,2,T} \\ \vec{H}_r^{-1,3,T} \\ \vec{H}_r^{-1,4,T} \end{pmatrix}$$

The derivation of changing the problem into the normal form (to be solved by DLT) is as follows:

$$0 = \overline{x_i} \times P H^{-1} \overline{X_i}$$

$$0 = \begin{pmatrix} x \\ y \\ w \end{pmatrix} \times P \begin{pmatrix} \overline{H_r}^{-1,1,T} \\ \overline{H_r}^{-1,2,T} \\ \overline{H_r}^{-1,3,T} \\ \overline{H_r}^{-1,4,T} \end{pmatrix} \overline{X_i}$$

$$0 = \begin{pmatrix} x \\ y \\ w \end{pmatrix} \times \begin{pmatrix} p_{1,1} & p_{1,2} & p_{1,3} & p_{1,4} \\ p_{2,1} & p_{2,2} & p_{2,3} & p_{2,4} \\ p_{3,1} & p_{3,2} & p_{3,3} & p_{3,4} \end{pmatrix} \begin{pmatrix} \overline{H_r}^{-1,1,T} \overline{X_i} \\ \overline{H_r}^{-1,2,T} \overline{X_i} \\ \overline{H_r}^{-1,3,T} \overline{X_i} \\ \overline{H_r}^{-1,4,T} \overline{X_i} \end{pmatrix}$$

$$0 = \begin{pmatrix} x \\ y \\ w \end{pmatrix} \times \begin{pmatrix} p_{1,1} & p_{1,2} & p_{1,3} & p_{1,4} \\ p_{2,1} & p_{2,2} & p_{2,3} & p_{2,4} \\ p_{3,1} & p_{3,2} & p_{3,3} & p_{3,4} \end{pmatrix} \begin{pmatrix} \overline{X_i}^T \overline{H_r}^{-1,1} \\ \overline{X_i}^T \overline{H_r}^{-1,2} \\ \overline{X_i}^T \overline{H_r}^{-1,3} \\ \overline{X_i}^T \overline{H_r}^{-1,4} \end{pmatrix}$$

$$0 = \begin{pmatrix} x \\ y \\ w \end{pmatrix} \times \begin{pmatrix} p_{1,1} \overline{X_i}^T \overline{H_r}^{-1,1} + p_{1,2} \overline{X_i}^T \overline{H_r}^{-1,2} + p_{1,3} \overline{X_i}^T \overline{H_r}^{-1,3} + p_{1,4} \overline{X_i}^T \overline{H_r}^{-1,4} \\ p_{2,1} \overline{X_i}^T \overline{H_r}^{-1,1} + p_{2,2} \overline{X_i}^T \overline{H_r}^{-1,2} + p_{2,3} \overline{X_i}^T \overline{H_r}^{-1,3} + p_{2,4} \overline{X_i}^T \overline{H_r}^{-1,4} \\ p_{3,1} \overline{X_i}^T \overline{H_r}^{-1,1} + p_{3,2} \overline{X_i}^T \overline{H_r}^{-1,2} + p_{3,3} \overline{X_i}^T \overline{H_r}^{-1,3} + p_{3,4} \overline{X_i}^T \overline{H_r}^{-1,4} \end{pmatrix}$$

$$0 = \begin{pmatrix} (yp_{3,1} - wp_{2,1}) \overline{X_i}^T & (yp_{3,2} - wp_{2,2}) \overline{X_i}^T & (yp_{3,3} - wp_{2,3}) \overline{X_i}^T & (yp_{3,4} - wp_{2,4}) \overline{X_i}^T \\ (wp_{1,1} - xp_{3,1}) \overline{X_i}^T & (wp_{1,2} - xp_{3,2}) \overline{X_i}^T & (wp_{1,3} - xp_{3,3}) \overline{X_i}^T & (wp_{1,4} - xp_{3,4}) \overline{X_i}^T \\ (xp_{2,1} - yp_{1,1}) \overline{X_i}^T & (xp_{2,2} - yp_{1,2}) \overline{X_i}^T & (xp_{2,3} - yp_{1,3}) \overline{X_i}^T & (xp_{2,4} - yp_{1,4}) \overline{X_i}^T \end{pmatrix} \begin{pmatrix} \overline{H_r}^{-1,1} \\ \overline{H_r}^{-1,2} \\ \overline{H_r}^{-1,3} \\ \overline{H_r}^{-1,4} \end{pmatrix}$$

where ***H*** is the vector that is the null space of the coefficient matrix and is the unknown H_{map} matrix in row major order.

This process can be repeated for each of the correspondences between 3D Euclidean, ground truth points and their 2D projection. The resulting coefficient matrix for each correspondence is then used as three rows in an *A* matrix. The nontrivial null space of the resulting *A* matrix, using the fact that $0=A\boldsymbol{H}$, is the ***H*** vector, and likewise the H_{map} in row major order.

C. RESULTS

Using these methods we were unable to find suitable mappings between projective and Euclidean space.

D. DISCUSSION

For both normal forms of the problem we solve using a DLT to produce the homography. However, we were not able to upgrade the geometry using these recommended methods.

With regard to the method using the equation "$\boldsymbol{X}_{\boldsymbol{Ei}}=H_{map}\boldsymbol{X}_{\boldsymbol{Pi}}$ *for all i*", we based our method on information from [31] as well as the description of DLT presented in [1]. According to the first reference, there are eight possible skew-symmetric forms of the three-dimensional homogenous vector differing in sign only. Only one of these possible skew-symmetric forms is the basis for creation of the A matrix, and without it we cannot perform a DLT. Theoretically it is possible to determine which form is desired but it is not clear if the form found is static for all points or if it is something that should be decided for each set of correspondences. Nevertheless, once we discovered the reference and tested it using our point cube, we were unable to transform the projective reconstruction to the Euclidean geometry using any combination of static and dynamic selection of the form of the skew symmetric matrix.

Turning attention to the method using the equation "$0 = \overline{x_i} \times PH^{-1}\overline{X_i}$," we did not have the same complexities in the mathematics, but we found that we were again unable to find the mapping between the geometric spaces. However, the reason, we now believe, is that we were attempting to use all correspondences between the ground truth points and their projective reconstruction counterparts. It may be that even when the correspondence between Euclidean and projective space is known, some of the 3D points in projective space could be outliers with respect to the estimated camera matrix. These special points could be classified as outliers by measuring their errors in reprojection (see experiment three) of the point by the same camera matrix used when triangulating its position.

Hence, if one were to define outliers and inliers in a projective reconstruction using a RANSAC or equivalent robust approach, and only then utilize this method on the inliers, the correct mapping between the two geometric spaces might be found. This notion is from the fact that in our unreported results, our projective reconstructions seemed to have one or two points that were arbitrarily placed as opposed to the others which seemed somewhat correct comparatively. When these oddly placed points were reprojected back onto the image plane, they had the maximum error of all errors in the epipolar geometry.

THIS PAGE INTENTIONALLY LEFT BLANK

XI. CONCLUSION

Overall our goal was to determine how camera motion and feature tracking effects the estimation of the epipolar geometry by way of the fundamental matrix. At various stages of the projective reconstruction we measured the errors in the current estimate of the epipolar geometry between two images using the equation $x'^{T}Fx=error$ where x'⇔x and F is the fundamental matrix relating the images. We provided a simplistic look at feature tracking comparing a multi-pass algorithm to the standard method of sequential feature tracking and saw that our multi-pass approach provided improved feature tracking performance.

As for camera motion, our expectation based upon various references was that the more translation in the camera's motion, the better the estimation of the epipolar geometry. However, the opposite may be true. The error in the epipolar geometry, on the average, was increased under projections with pure translational motion in the *xy*-plane between them. This means that when there is a high amount of parallax caused by this type of translational motion, we see, on average, an increase in the error in our estimate of the epipolar geometry. As the number of features that are being tracked increases, the camera motion seems to have little effect in initial estimates of the epipolar geometry, but a dramatic effect on the use of optimization especially when the camera's motion is unconstrained.

Furthermore, in general the camera motions translation and scaling, each with more parallax than rotation, were consistently more accurate in the matlab implementations we tested, and less accurate in the C/C++ implementations. This, we believe, is due to the implementation of SVD, but further investigation into this is needed.

In a system for performing SLAM, the robot will have a concept of what the expected motion of the camera should be due to the fact that it knows what commands are being issued as far as its movement. Returning to our initial scenario of small flying robots, relying on an expected motion could be problematic due to the size of the robots and the influence that air turbulence has on their motion. However, for a ground based robot, the knowledge of expected camera motion could bootstrap the vision based SLAM process by optimizing the overall process with respect to the results found in this thesis.

A. FUTURE WORK

There are two more steps necessary to complete a real-time or near real-time system for conducting SLAM. First, using the recommended algorithms for determining the epipolar geometry, we need to determine the metric motion of a camera in the real world and compare the estimates of the ground truth to develop a motion model for the sensor. Second, the algorithm should be extended to a multi sensor approach, e.g. use of occlusion as well as tracked features, to determine scene geometry to account for dynamic objects, shading changes, etc.

1. SLAM

The most obvious future work is the integration of our results into an implementation of vision based SLAM so that the system is optimized with respect to camera motion. However, there are various tasks that need to be addressed prior to this described below along with other ideas on methods of implementation.

2. Multi-Agent Vision Based SLAM

An implementation of vision based SLAM can be extended to multiple agents, sharing a single map as well as features for navigation and cooperative mapping. There are several ways to design such a system and they will only be briefly touched here. One method is that the robots are merely “camera men” who relay images to a central node, which processes all images from all robots and provides each robot with information about its location in a global map. Another method would be to have each robot develop its own map and have a central node, or master robot that combines the different maps into a global map. The whole system could be bootstrapped by lining the robots up so that they each see a similar view from different perspectives, and use a feature tracker between their individual views by the master robot to correlate the initial features.

3. Comparison of Parallax Geometry to Epipolar Geometry as a Means for Conducting SLAM

Parallax geometry differs from epipolar geometry in that the former focuses on the parallax shift between points over multiple images where the latter bypasses the depth information from parallax and focuses on the relationship between corresponding points and the epipolar geometry between two images. A look at parallax geometry as it relates to SLAM is warranted due to its ability to also conduct 3D reconstruction.

4. Overcoming Degeneracy when Tracking a Small Number of Features

In our experimentation with the initial errors in the epipolar geometry (experiment two and three), it would be interesting to selectively add and remove corresponding pairs to determine if the degeneracy conditions encountered in the A matrix (Chapter III) can be overcome for OpenCV. Essentially, we suggest creating a robust method like RANSAC which tests the random removal of points when small numbers of points are being tracked in an attempt to create an A matrix from the remaining points that has a rank ≥ 8.

5. Multiple Sensor Approach

There is no one panacea to retrieving depth from images; however, it has been shown that humans use different cues for estimating depth such as texture gradient, defocus, occlusion, and known object size[32]. The best model will come from a multi-dimensional approach to scene analysis. It is obvious that SFM can recreate a single scene offline with high accuracy and subjectively high quality. The problem is the non-real-time nature of the algorithm due to its iterative processes. Other cues are needed to bootstrap the process and these could come from research like Standford's use of a Markov Random Field (MRF) texture approach. It could be used as a first estimate of depth, and then fine tuned by using the epipolar geometry between the scenes. Furthermore, it may be possible to do online camera calibration using the MRF to bootstrap the process. The problem, as stated in the Literature Review, is that their approach requires training with ground truth on the type of scenes that will be used. Hence, it might be that SFM and the MRF work cooperatively to train the MRF which bootstraps the SFM approach.

Other clues to the scene structure could come from the use of Structure-from-Silhouette(SfS) algorithms where dynamic objects in the scene provide cues as to the depth of pixels by extracting the silhouette from the known background and inferring information about the scene structure based on occlusion of the silhouette. Again, this approach could be used to help bootstrap the SFM process.

6. Effect of Different Features and Feature Tracking Algorithms

There are various types of features and feature tracking algorithms available that can be used to create the corresponding points needed as input for a 3D reconstruction/3D scene analysis algorithm. The quality of the corresponding points is extremely important to downstream algorithms for producing a high quality estimate of epipolar geometry. Here, the quality of the points is measured by how well they are localized from the ground truth, and how many of the points are outliers versus inliers. Feature tracking is where the real world interfaces with the virtual world and how well the features can be localized and tracked as it relates to a robots ability to perform SLAM in real-time is the subject of a thesis in it of itself.

7. Detection of Outliers at Various Stages throughout Reconstruction

In general it seems that the noise that persists with each estimate could possibly be reduced through a robust approach from analysis of the errors at every stage of the projective reconstruction. For instance, one could attempt to use the resulting errors of the optimization of the initial estimate of the fundamental matrix with respect to the Sampson distance as a measure for determining outliers in order to identify points that may cause problems in the final projective reconstruction.

8. SLAM via Occlusion

Although this could be somewhat abstract, given a dynamic scene it may be possible to use the third-dimension recovery technique that leverages occlusion as described in Chapter III to perform SLAM in the right environment. It could be that there are objects in the environment that remain static with dynamic objects constantly encircling them, such as a turnstile in a subway terminal. If a robot were able to monitor a scene and use the occlusion method to extract 3D objects in the scene, then these objects could be used as features to localize upon.

9. Influence of the Expectation of Camera Motion on SLAM

A robot has an idea of how it moved between captured images. This could possibly bootstrap a vision based SLAM algorithm by providing a priori knowledge about its proposed motion to a decision tree as to which method of finding an estimate of the epipolar geometry. For instance, if it was tracking a large number of features, then it could decide not to perform optimization due to the performance of optimization from projections of the noisy cloud.

B. SUMMARY

The main thing to take away from this thesis is that SLAM can be done using only a monocular camera, and camera motion can have an effect on the overall performance of the SLAM system by introducing errors for different camera motions at various stages of a 3D reconstruction. The knowledge of when to expect errors can be used as an advantage when creating a system in so much as the expectation of a robot for a certain motion will allow it to decide what approach to use to conduct perform 3D reconstruction from the images at hand. There is a lot of work to be done still to allow robots and humans to use the same map (i.e., more than a sparse map) but the possibility of doing so seems within reach.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A: ERROR IN THE INITIAL EPIPOLAR GEOMETRY

This appendix is gives the results for Experiment 2, Chapter VI.

The results are initially categorized by the algorithm used in their calculation, and then further into tables based upon the type of projection of the two different synthetic scenes as described in Chapter IV. Finally, each row of the table shows which motions the camera underwent between projections, and the associated statistics for the trial.

- Average: Average of the absolute errors for each set of corresponding points to their respective epipolar line as defined by the estimated Fundamental matrix.
- Standard Deviation: The standard deviation associated with the average above.
- Max: The maximum error in the estimated epipolar geometry.
- L2 Norm: The square root of the sum of squares of all errors in the epipolar geometry.

Below is a sample table

Motion	**Average**	**Standard Deviation**	**Max**	**L2 Norm**
Pure Translation				
Pure Rotation				
Pure Scaling				
Unconstrained				
Motion the Camera Undertook Between Projections	**Average of the Absolute Error Values (L1)**	**Standard Deviation of the Absolute Error Values (L1)**	**Maximum Absolute Value of Dataset (Max)**	**Square Root of the Sum of Squares. (L2)**

Table 8. Sample "Results Table."

A. HARTLEY AND ZISSERMAN

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.0890	0.0852	0.2246	0.3379
Pure Rotation	3.9378×10^{-16}	2.4332×10^{-16}	8.8818×10^{-16}	1.2865×10^{-15}
Pure Scaling	3.4259×10^{-15}	3.7275×10^{-15}	8.5901×10^{-15}	1.3826×10^{-14}
Unconstrained	0.0469	0.0602	0.1883	0.2071

Table 9. Error in the Cube: Perfect Points dataset of the Hartley and Zisserman Implementation.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.0659	0.0215	0.0947	0.1948
Pure Rotation	3.4470×10^{-05}	2.1179×10^{-05}	6.8277×10^{-05}	0.0001
Pure Scaling	1.0723×10^{-05}	1.3335×10^{-07}	1.0939×10^{-05}	3.0332×10^{-05}
Unconstrained	6.4942×10^{-07}	2.4673×10^{-07}	1.0545×10^{-06}	1.9494×10^{-06}

Table 10. Error in the Cube: Noisy Points dataset of the Hartley and Zisserman Implementation.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	4.3668×10^{-16}	9.082×10^{-16}	7.1054×10^{-15}	1.4187×10^{-14}
Pure Rotation	4.8537×10^{-16}	6.2528×10^{-16}	3.5527×10^{-15}	1.1149×10^{-14}
Pure Scaling	5.5757×10^{-16}	9.7082×10^{-16}	9.2328×10^{-15}	1.5809×10^{-14}
Unconstrained	9.2440×10^{-16}	1.5878×10^{-15}	1.0658×10^{-14}	1.2731×10^{-15}

Table 11. Error in the Cloud: Perfect Points dataset of the Hartley and Zisserman Implementation.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.0104	0.0147	0.1032	0.2536
Pure Rotation	0.0227	0.0179	0.1072	0.4071
Pure Scaling	0.0193	0.01728	0.0968	0.3659
Unconstrained	0.0109	0.0119	0.0694	0.0143

Table 12. Error in the Cloud: Noisy Points dataset of the Hartley and Zisserman Implementation.

B. STRUCTURE AND MOTION TOOLKIT'S MAPSAC

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.2946	0.3273	0.7072	1.2019
Pure Rotation	6.6316	6.1598	13.5619	24.8480
Pure Scaling	0.2946	0.3273	0.7077	1.2018
Unconstrained	10.4553	19.5712	53.0680	58.9542

Table 13. Error in the Cube: Perfect Points dataset using SAM’s MAPSAC to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.2970	0.3200	0.6959	1.1926
Pure Rotation	32.7718	33.2742	78.3737	127.8363
Pure Scaling	0.2954	0.3233	0.7024	1.1958
Unconstrained	52.7753	22.8552	63.3145	69.2895

Table 14. Error in the Cube: Noisy Points dataset using SAM’s MAPSAC to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.3540	0.2717	1.1265	6.2892
Pure Rotation	10.5045	1.2427	13.0922	149.2128
Pure Scaling	0.4510	0.3389	1.3915	7.9512
Unconstrained	0.4690	0.3499	1.5320	8.2477

Table 15. Error in the Cloud: Perfect Points dataset using SAM’s MAPSAC to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.3446	0.2674	1.0958	6.1473
Pure Rotation	11.8744	1.4894	15.1915	168.8151
Pure Scaling	0.4527	0.3369	1.3190	7.9534
Unconstrained	0.4503	0.3377	1.7568	7.9325

Table 16. Error in the Cloud: Noisy Points dataset using SAM’s MAPSAC to estimate F.

C. STRUCTURE AND MOTION TOOLKIT'S LINEAR

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.2946	0.3273	0.7071	1.2019
Pure Rotation	6.6311	6.1594	13.5608	24.8462
Pure Scaling	0.2946	0.3273	0.7071	1.2019
Unconstrained	8.1252	11.8759	30.9807	38.9284

Table 17. Error in the Cube: Perfect Points dataset using SAM's Linear method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.2961	0.3191	0.6929	1.1891
Pure Rotation	6.6325	6.1683	13.5929	24.8648
Pure Scaling	0.2955	0.3233	0.7024	1.1958
Unconstrained	7.9532	11.5888	29.94141	38.0281

Table 18. Error in the Cube: Noisy Points dataset using SAM's Linear method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	1.1243	0.8365	4.1824	19.7500
Pure Rotation	9.6267	1.0920	11.8995	136.6676
Pure Scaling	0.0418	0.0449	0.2201	0.8640
Unconstrained	0.2961	0.3921	1.7491	6.9208

Table 19. Error in the Cloud: Perfect Points dataset using SAM's Linear method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	1.3665	1.0222	5.0593	24.05256
Pure Rotation	99.6904	18.6912	151.1104	1430.6886
Pure Scaling	0.0461	0.0605	0.2666	1.0715
Unconstrained	0.2633	0.2938	1.5571	5.5569

Table 20. Error in the Cloud: Noisy Points dataset using SAM's Linear method to estimate F.

D. STRUCTURE AND MOTION TOOLKIT'S MLESAC

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	6.4832	8.6357	20.8764	29.2964
Pure Rotation	13.4936	19.4036	57.6768	63.9696
Pure Scaling	13.5788	30.9954	88.6259	90.5542
Unconstrained	17.3862	29.7833	82.8834	92.8847

Table 21. Error in the Cube: Perfect Points dataset using SAM's MLESAC to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	17.8493	30.5242	84.2665	95.2411
Pure Rotation	273.8504	611.8370	1755.2320	1794.5373
Pure Scaling	0.0194	16.2957	37.5326	50.0964
Unconstrained	7.8707	10.7531	33.2657	36.1248

Table 22. Error in the Cube: Noisy Points dataset using SAM's MLESAC to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.8759	0.7519	3.7687	16.1333
Pure Rotation	3.4775	0.9055	6.9261	50.6839
Pure Scaling	0.3165	0.3481	1.7142	6.6280
Unconstrained	3.1063	0.5234	4.8303	44.4340

Table 23. Error in the Cloud: Perfect Points dataset using SAM's MLESAC to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.8759	0.7372	3.7687	16.1333
Pure Rotation	3.4775	0.9055	6.9261	50.6842
Pure Scaling	0.3187	0.3571	1.7924	6.7425
Unconstrained	2.6234	0.5681	4.6523	37.8608

Table 24. Error in the Cloud: Noisy Points dataset using SAM's MLESAC to estimate F.

E. OPENCV'S 8-POINT

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	Undefined	Undefined	Undefined	Undefined
Pure Rotation	Undefined	Undefined	Undefined	Undefined
Pure Scaling	$5.5951\text{x}10^{-07}$	$1.0342\text{x}10^{-06}$	$2.2352\text{x}10^{-06}$	$3.161\text{x}10^{-06}$
Unconstrained	0.3056	0.7046	2.0382	2.0549

Table 25. Error in the Cube: Perfect Points dataset using OpenCV’s 8-Point Algorithm to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	1.0451	0.4981	2.1977	3.2366
Pure Rotation	0.0016	0.0029	0.0087	0.0089
Pure Scaling	0.2436	0.0163	0.2700	0.6903
Unconstrained	$3.5627 \text{ x } 10^{-05}$	$1.6825 \text{ x } 10^{-05}$	$7.5659 \text{ x } 10^{-05}$	0.0001

Table 26. Error in the Cube: Noisy Points dataset using OpenCV’s 8-Point Algorithm to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	1.4490	0.8235	3.6328	22.0608
Pure Rotation	0.0000	2.5436	12.0000	43.0349
Pure Scaling	5.3105×10^{-05}	5.1503×10^{-05}	0.0004	0.0008
Unconstrained	4.2422×10^{-07}	1.0326×10^{-07}	1.0729×10^{-06}	6.4960×10^{-06}

Table 27. Error in the Cloud: Perfect Points dataset using OpenCV's 8-Point Algorithm to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.7198	1.5318	10.7654	26.5972
Pure Rotation	0.0269	0.0171	0.0990	0.3881
Pure Scaling	2.1763	5.2459	28.5684	111.1578
Unconstrained	0.0008	0.0025	0.0143	0.04690

Table 28. Error in the Cloud: Noisy Points dataset using OpenCV's 8-Point Algorithm to estimate F.

F. OPENCV'S RANSAC

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	Undefined	Undefined	Undefined	Undefined
Pure Rotation	Undefined	Undefined	Undefined	Undefined
Pure Scaling	5.5951×10^{-07}	1.0342×10^{-06}	2.2352×10^{-06}	3.1610×10^{-06}
Unconstrained	0.3056	0.7046	2.0382	2.0549

Table 29. Error in the Cube: Perfect Points dataset using OpenCV’s RANSAC Algorithm to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	1.0451	0.4982	2.1977	3.2366
Pure Rotation	0.0016	0.0029	0.0087	0.0089
Pure Scaling	0.2436	0.0163	0.2700	0.6903
Unconstrained	3.5627×10^{-05}	1.6825×10^{-05}	7.5659×10^{-05}	0.0001

Table 30. Error in the Cube: Noisy Points dataset using OpenCV’s RANSAC Algorithm to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	1.4490	0.8235	3.6328	22.0608
Pure Rotation	0.0000	2.5436	12.0000	43.0349
Pure Scaling	5.3105×10^{-06}	5.1503×10^{-05}	0.0004	0.0008
Unconstrained	4.2422×10^{-07}	1.0326×10^{-07}	1.0729×10^{-06}	6.4960×10^{-06}

Table 31. Error in the Cloud: Perfect Points dataset using OpenCV's RANSAC Algorithm to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.7198	1.5319	10.7654	26.5972
Pure Rotation	0.0269	0.0171	0.0989	0.3881
Pure Scaling	2.1763	5.2459	28.5684	111.1578
Unconstrained	0.0008	0.0025	0.0143	0.0469

Table 32. Error in the Cloud: Noisy Points dataset using OpenCV's RANSAC Algorithm to estimate F.

G. OPENCV'S LMEDS

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	Undefined	Undefined	Undefined	Undefined
Pure Rotation	Undefined	Undefined	Undefined	Undefined
Pure Scaling	5.5951×10^{-07}	1.0342×10^{-06}	2.2352×10^{-06}	3.1610×10^{-06}
Unconstrained	0.0508	0.0823	0.2469	0.2610

Table 33. Error in the Cube: Perfect Points dataset using OpenCV’s LMEDS to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.0020	0.0055	0.0157	0.0157
Pure Rotation	0.0016	0.0029	0.0087	0.0089
Pure Scaling	0.2436	0.0163	0.2700	0.6903
Unconstrained	4.6349×10^{-06}	9.2013×10^{-06}	2.6756×10^{-5}	2.7650×10^{-05}

Table 34. Error in the Cube: Noisy Points dataset using OpenCV’s LMEDS algorithm to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	1.4490	0.8235	3.6328	22.0608
Pure Rotation	0.0000	2.5436	12.0000	43.0349
Pure Scaling	5.3105×10^{-06}	5.1503×10^{-05}	0.0004	0.0008
Unconstrained	$4.2422 \text{ X } 10^{-07}$	1.0326×10^{-07}	1.0729×10^{-06}	6.4960×10^{-06}

Table 35. Error in the Cloud: Perfect Points dataset using LMEDS Algorithm to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.7198	1.5318	10.7654	26.5972
Pure Rotation	0.0269	0.0171	0.0990	0.3881
Pure Scaling	2.1763	5.2459	28.5684	111.1578
Unconstrained	0.0008	0.0025	0.0143	0.0469

Table 36. Error in the Cloud: Noisy Points dataset using OpenCV's LMEDS Algorithm to estimate F.

H. MA'S 8-POINT

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.1004	0.1634	0.3982	0.5171
Pure Rotation	2.8554×10^{-15}	5.5316×10^{-15}	1.5543×10^{-14}	1.6716×10^{-14}
Pure Scaling	5.1855×10^{-16}	6.0415×10^{-16}	1.6495×10^{-15}	2.1694×10^{-15}
Unconstrained	4.7386×10^{-14}	1.1216×10^{-13}	3.2152×10^{-13}	3.2562×10^{-13}

Table 37. Error in the Cube: Perfect Points dataset of Ma's 8-Point Implementation.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.0059	0.0080	0.0195	0.0271
Pure Rotation	0.0223	0.0473	0.1381	0.1402
Pure Scaling	0.0367	0.0429	0.1032	0.1537
Unconstrained	0.1612	0.3625	1.0362	1.0619

Table 38. Error in the Cube: Noisy Points dataset of Ma's 8-Point Implementation.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	5.5988×10^{-16}	1.1012×10^{-15}	8.4377×10^{-15}	1.7392×10^{-14}
Pure Rotation	8.0734×10^{-15}	6.7482×10^{-15}	$3.9080 \times 10^{--14}$	1.4828×10^{-13}
Pure Scaling	8.4103×10^{-16}	1.5398×10^{-15}	9.8254×10^{-15}	2.4703×10^{-14}
Unconstrained	1.4457×10^{-15}	1.3367×10^{-15}	1.0658×10^{-14}	2.7743×10^{-14}

Table 39. Error in the Cloud: Perfect Points dataset of Ma's 8-Point Implementation.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.0203	0.0244	0.1459	0.4472
Pure Rotation	0.0826	0.0678	0.2989	1.5051
Pure Scaling	0.0196	0.0205	0.1306	0.3994
Unconstrained	0.4636	0.1353	1.0982	6.8113

Table 40. Error in the Cloud: Noisy Points dataset of Ma's 8-Point Implementation.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B: REPROJECTION ERROR

This appendix gives the results for Experiment 3, Chapter VII.

The results are initially categorized by the algorithm used in their calculation, and then further into tables based upon the synthetic scene the original point correspondences were used from, as well as the camera used to reproject the 3D points of the projective reconstruction. Finally, each row of the table shows which motions the camera underwent between projections to find the original point correspondences.

- Average: Average of the absolute errors for each set of corresponding points to their respective epipolar line as defined by the estimated Fundamental matrix.
- Standard Deviation: The standard deviation associated with the average above.
- Max: The maximum error in the estimated epipolar geometry.
- L2 Norm: The square root of the sum of squares of all errors in the epipolar geometry.

Below is a sample table

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation				
Pure Rotation				
Pure Scaling				
Unconstrained				
Motion the Camera Undertook Between Projections	**Average of the Absolute Error Values (L1)**	**Standard Deviation of the Absolute Error Values (L1)**	**Maximum Absolute Value of Dataset (Max)**	**Square Root of the Sum of Squares. (L2)**

Table 41. Sample "Results Table."

A. STRUCTURE AND MOTION TOOLKIT'S LINEAR METHOD

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	$6.3294 \times 10^{+14}$	$2.1198 \times 10^{+14}$	$1.0016 \times 10^{+15}$	$1.8760 \times 10^{+15}$
Pure Rotation	$2.4574 \times 10^{+13}$	$4.0001 \times 10^{+13}$	$1.1621 \times 10^{+14}$	$1.2663 \times 10^{+14}$
Pure Scaling	$1.9531 \times 10^{+13}$	$2.1919 \times 10^{+13}$	$6.5431 \times 10^{+13}$	$8.0092 \times 10^{+13}$
Unconstrained	$2.3069 \times 10^{+14}$	$2.4019 \times 10^{+14}$	$5.2308 \times 10^{+14}$	$9.1081 \times 10^{+14}$

Table 42. Image 1 reprojection error of the Cube: Perfect Points dataset using SAM’s Linear method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	1.8070	0.5993	2.8190	5.3511
Pure Rotation	9.4220	3.6477	14.2406	28.3431
Pure Scaling	0.5859	0.1243	0.7049	1.6894
Unconstrained	20.1428	21.2052	51.6758	79.9593

Table 43. Image 2 reprojection error of the Cube: Perfect Points dataset using SAM’s Linear method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	$3.994 \times 10^{+13}$	$1.0512 \times 10^{+13}$	$5.5222 \times 10^{+13}$	$1.1634 \times 10^{+14}$
Pure Rotation	$3.0974 \times 10^{+13}$	$5.7263 \times 10^{+13}$	$1.7266 \times 10^{+14}$	$1.7501 \times 10^{+14}$
Pure Scaling	$1.0195 \times 10^{+13}$	$2.3134 \times 10^{+13}$	$6.7406 \times 10^{+13}$	$6.7659 \times 10^{+13}$
Unconstrained	$2.8762 \times 10^{+14}$	$3.9043 \times 10^{+14}$	$1.2133 \times 10^{+15}$	$1.3148 \times 10^{+15}$

Table 44. Image 1 reprojection error of the Cube: Noisy Points dataset using SAM's Linear method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	1.9645	0.9311	4.0048	6.0781
Pure Rotation	9.2987	3.8201	14.4085	28.1758
Pure Scaling	0.5911	0.1263	0.7362	1.7050
Unconstrained	19.6502	20.5311	49.7749	77.7157

Table 45. Image 2 reprojection error of the Cube: Noisy Points dataset using SAM's Linear method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	4.2435	0.6220	5.9945	135.6223
Pure Rotation	56.1279	1.3538	59.1855	1775.4363
Pure Scaling	0.1297	0.0488	0.2495	4.3809
Unconstrained	0.9307	0.4276	1.9147	32.3867

Table 46. Image 1 reprojection error of the Cloud: Perfect Points dataset using SAM's Linear method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	4.2932	0.6387	6.1487	137.2569
Pure Rotation	11.1921	0.9701	13.9910	355.2504
Pure Scaling	0.7795	0.2901	1.5470	26.3019
Unconstrained	0.8360	0.3598	1.8267	28.7799

Table 47. Image 2 reprojection error of the Cloud: Perfect Points dataset using SAM's Linear method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	4.2485	0.6216	6.0000	135.7799
Pure Rotation	14.1762	1.0084	16.8533	449.4235
Pure Scaling	0.1360	0.0546	0.2649	4.6340
Unconstrained	0.7775	0.3047	1.5786	26.4063

Table 48. Image 1 reprojection error of the Cloud: Noisy Points dataset using SAM's Linear method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	4.2960	0.6507	6.2657	137.3993
Pure Rotation	11.2048	0.9734	13.9884	355.6615
Pure Scaling	0.7736	0.2885	1.5198	26.1073
Unconstrained	0.7723	0.3223	1.6273	26.4623

Table 49. Image 2 reprojection error of the Cloud: Noisy Points dataset using SAM's Linear method to estimate F.

B. STRUCTURE AND MOTION TOOLKIT'S MAPSAC METHOD

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	36275.1276	916.6236	37262.0220	102630.3933
Pure Rotation	135253.6682	7539.9897	146553.3727	383075.1141
Pure Scaling	165.1283	0.2888	165.6160	467.2310
Unconstrained	64.4759	22.5502	115.5939	192.1036

Table 50. Image 1 reprojection error of the Cube: Perfect Points dataset using SAM's MAPSAC method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	1.8085	0.5975	2.8174	5.3538
Pure Rotation	9.4100	3.6425	14.2404	28.3066
Pure Scaling	0.5881	0.1258	0.7100	1.6964
Unconstrained	20.2163	21.2896	51.8943	80.2640

Table 51. Image 2 reprojection error of the Cube: Perfect Points dataset using SAM's MAPSAC method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	489.5682	3.5270	494.1827	1384.7393
Pure Rotation	12.6635	5.2505	18.8699	38.4174
Pure Scaling	11970.5303	4915.4639	16665.9573	36269.5638
Unconstrained	52.7753	27.9633	108.6103	166.5999

Table 52. Image 1 reprojection error of the Cube: Noisy Points dataset using SAM's MAPSAC method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	1.8068	0.5850	2.8047	5.3395
Pure Rotation	9.4377	3.6826	14.3281	28.4165
Pure Scaling	.5909	0.1270	0.7351	1.70487
Unconstrained	19.6936	20.6001	50.1254	77.9311

Table 53. Image 2 reprojection error of the Cube: Noisy Points dataset using SAM's MAPSAC method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	10.2618	0.6032	11.4039	325.0659
Pure Rotation	146357.5144	16462.6138	176143.8011	4657388.61
Pure Scaling	6.7457	0.0966	6.9101	213.3405
Unconstrained	16.7193	0.5840	17.8174	529.0329

Table 54. Image 1 reprojection error of the Cloud: Perfect Points dataset using SAM’s MAPSAC method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	4.2252	0.6405	6.0850	135.1385
Pure Rotation	11.1921	0.9701	13.9910	355.2504
Pure Scaling	0.8066	1.0073	31.2771	40.7947
Unconstrained	0.8278	0.8814	18.9574	38.2273

Table 55. Image 2 reprojection error of the Cloud: Perfect Points dataset using SAM’s MAPSAC method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	9.1627	0.602	10.3055	290.3745
Pure Rotation	88.5006	0.9619	90.9962	2798.8003
Pure Scaling	11.7747	0.4003	11.9734	372.5622
Unconstrained	3.7205	0.5799	4.8365	119.0720

Table 56. Image 1 reprojection error of the Cloud: Noisy Points dataset using SAM's MAPSAC method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	4.2961	0.6502	6.2644	137.3986
Pure Rotation	11.1949	0.9728	13.9812	355.3474
Pure Scaling	0.7738	0.2888	1.5242	26.1164
Unconstrained	0.7721	0.3222	1.6261	26.4533

Table 57. Image 2 reprojection error of the Cloud: Noisy Points dataset using SAM's MAPSAC method to estimate F.

C. HARTLEY AND ZISSERMAN IMPLEMENTATION

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	1.0050	0.9367	2.7027	3.7713
Pure Rotation	3.0298×10^{-12}	5.7272×10^{-12}	1.6884×10^{-11}	2.9592×10^{-06}
Pure Scaling	1.0132×10^{-13}	1.4986×10^{-13}	4.3825×10^{-13}	5.0119×10^{-07}
Unconstrained	0.6135	0.4558	1.6361	2.3527

Table 58. Image 1 reprojection error of the Cube: Perfect Points dataset using Hartley and Zisserman method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.0902	0.0984	0.2629	0.3644
Pure Rotation	1.1701×10^{-13}	2.8458×10^{-13}	8.1576×10^{-13}	8.2246×10^{-13}
Pure Scaling	2.5299×10^{-13}	4.9432×10^{-13}	1.4515×10^{-12}	1.4908×10^{-12}
Unconstrained	0.7366	0.9854	2.8949	3.3374

Table 59. Image 2 reprojection error of the Cube: Perfect Points dataset using Hartley and Zisserman method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	1.1988	0.9519	2.6038	4.2236
Pure Rotation	0.0004	0.0004	0.0010	0.0015
Pure Scaling	7.8955×10^{-08}	8.6428×10^{-08}	2.3282×10^{-07}	3.1962×10^{-07}
Unconstrained	2.0897×10^{-06}	1.5232×10^{-06}	4.0899×10^{-06}	7.1537×10^{-06}

Table 60. Image 1 reprojection error of the Cube: Noisy Points dataset using Hartley and Zisserman method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	3.0112	6.2841	18.3802	18.6807
Pure Rotation	0.0081	0.0070	0.0179	0.0294
Pure Scaling	0.0009	0.0012	0.0034	0.0041
Unconstrained	4.9038×10^{-05}	3.4657×10^{-05}	9.7034×10^{-05}	0.0002

Table 61. Image 2 reprojection error of the Cube: Noisy Points dataset using Hartley and Zisserman method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.2769	0.5781	3.6593	9.0459
Pure Rotation	3.3385×10^{-14}	2.9254×10^{-14}	1.6007×10^{-13}	6.2706×10^{-13}
Pure Scaling	6.1222×10^{-16}	6.1186×10^{-16}	3.5872×10^{-15}	1.2226×10^{-14}
Unconstrained	2.7846×10^{-14}	2.4891×10^{-14}	1.4569×10^{-13}	5.2761×10^{-13}

Table 62. Image 1 reprojection error of the Cloud: Perfect Points dataset using Hartley and Zisserman method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	8.6169×10^{-17}	6.0002×10^{-16}	5.9746×10^{-15}	8.5516×10^{-15}
Pure Rotation	1.4044×10^{-15}	3.7381×10^{-15}	2.1316×10^{-14}	5.6349×10^{-14}
Pure Scaling	3.8045×10^{-16}	6.3778×10^{-16}	5.0243×10^{-15}	1.0483×10^{-14}
Unconstrained	5.3979×10^{-17}	6.9418×10^{-16}	9.7966×10^{-15}	9.8223×10^{-15}

Table 63. Image 2 reprojection error of the Cloud: Perfect Points dataset using Hartley and Zisserman method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.076	0.0107	0.0738	0.1850
Pure Rotation	0.0179	0.0141	0.0778	0.3224
Pure Scaling	0.0099	0.0094	0.0707	0.1930
Unconstrained	0.0064	0.0070	0.0410	0.1342

Table 64. Image 1 reprojection error of the Cloud: Noisy Points dataset using Hartley and Zisserman method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	4.7646×10^{-07}	6.5768×10^{-07}	4.4761×10^{-06}	$1.1467 \text{ X } 10^{-05}$
Pure Rotation	1.5188×10^{-05}	1.1999×10^{-05}	6.6200×10^{-05}	0.0003
Pure Scaling	0.0002	0.0002	0.0016	0.0042
Unconstrained	3.6642×10^{-06}	4.042×10^{-06}	2.4164×10^{-05}	7.7048×10^{-05}

Table 65. Image 2 reprojection error of the Cloud: Noisy Points dataset using Hartley and Zisserman method to estimate F.

D. OPENCV'S 8-POINT METHOD

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	Undefined	Undefined	Undefined	Undefined
Pure Rotation	Undefined	Undefined	Undefined	Undefined
Pure Scaling	5.0679×10^{-06}	5.2754×10^{-06}	1.0271×10^{-05}	2.0007×10^{-05}
Unconstrained	0.0741	0.0838	0.2126	0.3049

Table 66. Image 1 reprojection error of the Cube: Perfect Points dataset using OpenCV's 8 Point method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	Undefined	Undefined	Undefined	Undefined
Pure Rotation	Undefined	Undefined	Undefined	Undefined
Pure Scaling	6.0242×10^{-05}	0.0001	0.0002	0.0003
Unconstrained	0.0006	0.0013	0.0039	0.0040

Table 67. Image 2 reprojection error of the Cube: Perfect Points dataset using OpenCV's 8 Point method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	4.3826	4.9713	16.0069	18.0736
Pure Rotation	0.0085	0.0073	0.0205	0.0308
Pure Scaling	0.0021	0.0019	0.0061	0.0077
Unconstrained	5.3935×10^{-05}	3.4041×10^{-05}	9.8601×10^{-05}	0.0002

Table 68. Image 1 reprojection error of the Cube: Noisy Points dataset using OpenCV's 8 Point method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.0051	0.0053	0.0148	0.0200
Pure Rotation	0.0004	0.0003	0.0008	0.0013
Pure Scaling	1.9845×10^{-06}	2.5096×10^{-06}	6.7435×10^{-06}	8.6945×10^{-06}
Unconstrained	5.0119×10^{-05}	0.0001	0.0004	0.0004

Table 69. Image 2 reprojection error of the Cube: Noisy Points dataset using OpenCV's 8 Point method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.6232	0.5740	2.9356	11.1643
Pure Rotation	Undefined	Undefined	Undefined	Undefined
Pure Scaling	0.0002	0.0015	0.0182	0.0215
Unconstrained	1.0168×10^{-05}	5.3783×10^{-06}	2.4844×10^{-05}	0.0002

Table 70. Image 1 reprojection error of the Cloud: Perfect Points dataset using OpenCV's 8 Point method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	8.7498×10^{-08}	9.5934×10^{-08}	5.9605×10^{-07}	1.8338×10^{-06}
Pure Rotation	Undefined	Undefined	Undefined	Undefined
Pure Scaling	2.0040×10^{-05}	1.5504×10^{-05}	7.2092×10^{-05}	0.0004
Unconstrained	6.2806×10^{-07}	7.7382×10^{-07}	6.7108×10^{-06}	1.4073×10^{-05}

Table 71. Image 2 reprojection error of the Cloud: Perfect Points dataset using OpenCV's 8 Point method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.0776	0.1905	1.2771	2.9014
Pure Rotation	0.0179	0.0143	0.0777	0.3242
Pure Scaling	0.0161	0.0319	0.2988	0.5039
Unconstrained	0.0063	0.0070	0.0415	0.1327

Table 72. Image 1 reprojection error of the Cloud: Noisy Points dataset using OpenCV's 8 Point method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	1.6837×10^{-07}	2.5994×10^{-07}	2.3223×10^{-06}	4.3721×10^{-06}
Pure Rotation	1.7222×10^{-05}	1.3845×10^{-05}	7.5281×10^{-05}	0.0003
Pure Scaling	6.0955×10^{-08}	6.1649×10^{-08}	3.6256×10^{-07}	1.2245×10^{-06}
Unconstrained	8.7269×10^{-05}	9.6715×10^{-05}	0.0006	0.0018

Table 73. Image 2 reprojection error of the Cloud: Noisy Points dataset using OpenCV's 8 Point method to estimate F.

E. OPENCV'S RANSAC METHOD

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	Undefined	Undefined	Undefined	Undefined
Pure Rotation	Undefined	Undefined	Undefined	Undefined
Pure Scaling	5.0679×10^{-06}	5.2754×10^{-06}	1.0271×10^{-05}	2.0007×10^{-05}
Unconstrained	0.0741	0.0838	0.2126	0.3049

Table 74. Image 1 reprojection error of the Cube: Perfect Points dataset using OpenCV's RANSAC method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	Undefined	Undefined	Undefined	Undefined
Pure Rotation	Undefined	Undefined	Undefined	Undefined
Pure Scaling	6.0242×10^{-05}	0.0001	0.0002	0.0003
Unconstrained	0.0006	0.0013	0.0039	0.0040

Table 75. Image 2 reprojection error of the Cube: Perfect Points dataset using OpenCV's RANSAC method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	4.3826	4.9713	16.0069	18.0736
Pure Rotation	0.0085	0.0073	0.0205	0.0308
Pure Scaling	0.0021	0.0019	0.0061	0.0077
Unconstrained	5.3935×10^{-05}	3.4041×10^{-05}	9.8601×10^{-05}	0.0002

Table 76. Image 1 reprojection error of the Cube: Noisy Points dataset using OpenCV's RANSAC method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.0051	0.0053	0.0148	0.0200
Pure Rotation	0.0004	0.0003	0.0008	0.0013
Pure Scaling	1.9845×10^{-06}	2.5096×10^{-06}	6.7435×10^{-06}	8.6945×10^{-06}
Unconstrained	5.0119×10^{-05}	0.0001	0.0004	0.0004

Table 77. Image 2 reprojection error of the Cube: Noisy Points dataset using OpenCV's RANSAC method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.6232	0.5740	2.9356	11.1643
Pure Rotation	Undefined	Undefined	Undefined	Undefined
Pure Scaling	0.0002	0.0015	0.0182	0.0215
Unconstrained	1.0168×10^{-05}	5.3783×10^{-06}	2.4844×10^{-05}	0.0002

Table 78. Image 1 reprojection error of the Cloud: Perfect Points dataset using OpenCV's RANSAC method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	8.7498×10^{-08}	9.5934×10^{-08}	5.9605×10^{-07}	1.8338×10^{-06}
Pure Rotation	Undefined	Undefined	Undefined	Undefined
Pure Scaling	2.0040×10^{-05}	1.5504×10^{-05}	7.2092×10^{-05}	0.0004
Unconstrained	6.2806×10^{-07}	7.7382×10^{-07}	6.7108×10^{-06}	1.4073×10^{-05}

Table 79. Image 2 reprojection error of the Cloud: Perfect Points dataset using OpenCV's RANSAC method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.0774	0.1905	1.2771	2.9014
Pure Rotation	0.0179	0.0143	0.0777	0.32428
Pure Scaling	0.0161	0.0319	0.2988	0.5039
Unconstrained	0.0063	0.0070	0.0415	0.1327

Table 80. Image 1 reprojection error of the Cloud: Noisy Points dataset using OpenCV's RANSAC method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	1.6836×10^{-07}	2.5994×10^{-07}	2.3223×10^{-06}	4.3721×10^{-06}
Pure Rotation	1.7222×10^{-05}	1.3845×10^{-05}	7.5281×10^{-05}	0.0003
Pure Scaling	6.0955×10^{-08}	6.1649×10^{-08}	3.6256×10^{-07}	1.2245×10^{-06}
Unconstrained	8.7269×10^{-05}	9.6715×10^{-05}	0.0006	0.0018

Table 81. Image 2 reprojection error of the Cloud: Noisy Points dataset using OpenCV's RANSAC method to estimate F.

F. OPENCV'S LMEDS METHOD

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	Undefined	Undefined	Undefined	Undefined
Pure Rotation	Undefined	Undefined	Undefined	Undefined
Pure Scaling	5.0679×10^{-06}	5.2754×10^{-06}	1.0271×10^{-05}	2.0007×10^{-05}
Unconstrained	0.0741	0.0838	0.2126	0.3049

Table 82. Image 1 reprojection error of the Cube: Perfect Points dataset using OpenCV's LMEDS method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	Undefined	Undefined	Undefined	Undefined
Pure Rotation	Undefined	Undefined	Undefined	Undefined
Pure Scaling	6.0242×10^{-05}	0.0001	0.0002	0.0003
Unconstrained	0.0006	0.0013	0.0039	0.0040

Table 83. Image 2 reprojection error of the Cube: Perfect Points dataset using OpenCV's LMEDS method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	1.2550	3.5488	10.0379	10.0379
Pure Rotation	0.0085	0.0073	0.0205	0.0308
Pure Scaling	0.0021	0.0019	0.0061	0.0077
Unconstrained	0.0215	0.0608	0.1721	0.1721

Table 84. Image 1 reprojection error of the Cube: Noisy Points dataset using OpenCV's LMEDS method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.0002	0.0005	0.0013	0.0013
Pure Rotation	0.0004	0.0003	0.0008	0.0013
Pure Scaling	1.9845×10^{-06}	2.5096×10^{-06}	6.7435×10^{-06}	8.6945×10^{-06}
Unconstrained	0.0008	0.0021	0.0060	0.0060

Table 85. Image 2 reprojection error of the Cube: Noisy Points dataset using OpenCV's LMEDS method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.6232	0.5740	2.9356	11.9682
Pure Rotation	Undefined	Undefined	Undefined	Undefined
Pure Scaling	0.0002	0.00153	0.0182	0.0215
Unconstrained	1.0168×10^{-05}	5.3783×10^{-06}	2.4844×10^{-05}	0.0002

Table 86. Image 1 reprojection error of the Cloud: Perfect Points dataset using OpenCV's LMEDS method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	8.7498×10^{-08}	9.5934×10^{-08}	5.9605×10^{-07}	1.8338×10^{-06}
Pure Rotation	Undefined	Undefined	Undefined	Undefined
Pure Scaling	2.0040×10^{-05}	1.5504×10^{-05}	7.2092×10^{-05}	0.0004
Unconstrained	6.2806×10^{-07}	7.7382×10^{-07}	6.7108×10^{-06}	1.4073×10^{-05}

Table 87. Image 2 reprojection error of the Cloud: Perfect Points dataset using OpenCV's LMEDS method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.0776	0.1904	1.2772	2.9014
Pure Rotation	0.0179	0.0143	0.0777	0.3242
Pure Scaling	0.0161	0.0319	0.2988	0.5039
Unconstrained	0.0063	0.0070	0.0415	0.1327

Table 88. Image 1 reprojection error of the Cloud: Noisy Points dataset using OpenCV's LMEDS method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	1.6837×10^{-07}	2.5994×10^{-07}	2.3223×10^{-06}	4.3721×10^{-06}
Pure Rotation	1.7222×10^{-05}	1.3845×10^{-05}	7.5281×10^{-05}	0.0003
Pure Scaling	6.0955×10^{-08}	6.1649×10^{-08}	3.6256×10^{-07}	1.2245×10^{-06}
Unconstrained	8.7269×10^{-05}	9.6715×10^{-05}	0.0006	0.0018

Table 89. Image 2 reprojection error of the Cloud: Noisy Points dataset using OpenCV's LMEDS method to estimate F.

G. MA'S 8-POINT IMPLEMENTATION

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.0220	0.0317	0.0906	0.2542
Pure Rotation	2.4101×10^{-13}	4.3501×10^{-13}	1.1937×10^{-12}	8.2220×10^{-07}
Pure Scaling	6.3705×10^{-14}	9.9718×10^{-14}	2.2167×10^{-13}	4.0426×10^{-07}
Unconstrained	6.0988×10^{-14}	9.4174×10^{-14}	2.3093×10^{-13}	3.9391×10^{-07}

Table 90. Image 1 reprojection error of the Cube: Perfect Points dataset using Ma's 8 Point method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	1.11498	1.4815	3.8498	5.0309
Pure Rotation	7.4034×10^{-14}	1.6493×10^{-13}	4.6966×10^{-13}	4.8401×10^{-13}
Pure Scaling	2.6966×10^{-13}	2.9776×10^{-13}	7.7375×10^{-13}	1.0965×10^{-12}
Unconstrained	1.4192×10^{-12}	3.1978×10^{-12}	9.1893×10^{-12}	9.3646×10^{-12}

Table 91. Image 2 reprojection error of the Cube: Perfect Points dataset using Ma's 8 Point method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.0722	0.0953	0.2175	0.3244
Pure Rotation	0.0104	0.0057	0.0229	0.0331
Pure Scaling	8.3727 x 10^{-05}	8.2541 x 10^{-05}	0.0002	0.0003
Unconstrained	0.1634	0.2861	0.7863	0.8869

Table 92. Image 1 reprojection error of the Cube: Noisy Points dataset using Ma's 8 Point method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.0216	0.0314	0.0847	0.1030
Pure Rotation	0.2640	0.1289	0.5317	0.8208
Pure Scaling	0.2135	0.1736	0.4511	0.7586
Unconstrained	5.4394	12.8772	36.9641	37.3826

Table 93. Image 2 reprojection error of the Cube: Noisy Points dataset using Ma's 8 Point method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.1029	0.2772	2.7441	4.1725
Pure Rotation	2.2347×10^{-14}	1.4529×10^{-14}	5.9064×10^{-14}	3.7667×10^{-13}
Pure Scaling	3.9402×10^{-16}	4.6176×10^{-16}	2.6738×10^{-15}	8.5722×10^{-15}
Unconstrained	2.0780×10^{-14}	1.6839×10^{-14}	8.1229×10^{-14}	3.7787×10^{-13}

Table 94. Image 1 reprojection error of the Cloud: Perfect Points dataset using Ma's 8 Point method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	1.1185×10^{-16}	6.1737×10^{-16}	5.9746×10^{-15}	8.8515×10^{-15}
Pure Rotation	2.7978×10^{-16}	1.5463×10^{-15}	1.2435×10^{-14}	2.2169×10^{-14}
Pure Scaling	4.4169×10^{-16}	9.4802×10^{-16}	7.2836×10^{-15}	1.4760×10^{-14}
Unconstrained	8.8818×10^{-18}	9.0154×10^{-17}	1.0547×10^{-15}	1.2780×10^{-14}

Table 95. Image 2 reprojection error of the Cloud: Perfect Points dataset using Ma's 8 Point method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	0.0191	0.0226	0.1358	0.4181
Pure Rotation	0.0870	0.0742	0.3661	1.6148
Pure Scaling	0.0130	0.0138	0.0991	0.2678
Unconstrained	0.3289	0.0991	0.7700	4.8563

Table 96. Image 1 reprojection error of the Cloud: Noisy Points dataset using Ma's 8 Point method to estimate F.

Motion	Average	Standard Deviation	Max	L2 Norm
Pure Translation	2.5256×10^{-06}	2.9948×10^{-06}	1.7285×10^{-05}	5.5322×10^{-05}
Pure Rotation	9.7192×10^{-05}	8.1303×10^{-05}	0.0004	0.0018
Pure Scaling	0.0003	0.0003	0.0020	0.0058
Unconstrained	2.3612×10^{-05}	7.2331×10^{-06}	5.5620×10^{-05}	0.0003

Table 97. Image 2 reprojection error of the Cloud: Noisy Points dataset using Ma's 8 Point method to estimate F.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C: SAMPSON DISTANCE OPTIMIZATION RESULTS

Motion	Average	Standard Deviation	Max	L2 Norm
Translation: Before	0.0890	0.0852	0.2246	0.3379
Translation: After	0.0002	6.7211×10^{-05}	0.0002	0.0005
Rotation: Before	0.3978×10^{-16}	2.4332×10^{-16}	8.8818×10^{-16}	1.2865×10^{-15}
Rotation: After	2.6999×10^{-13}	4.0282×10^{-13}	1.1053×10^{-12}	1.3095×10^{-12}
Scale: Before	3.4259×10^{-15}	3.7275×10^{-15}	8.5901×10^{-15}	1.3826×10^{-14}
Scale: After	3.4259×10^{-15}	3.7275×10^{-15}	8.5901×10^{-15}	1.3826×10^{-14}
Unconstrained: Before	0.0469	0.0602	0.1883	0.2071
Unconstrained: After	3.8643×10^{-05}	2.2124×10^{-05}	6.4622×10^{-05}	0.0001

Table 98. Sampson Distance Optimization: Noiseless Cube Projections.

Comparison of the errors in the epipolar geometry before and after the minimization of the Sampson Distance.

Motion	Average	Standard Deviation	Max	L2 Norm
Translation: Before	0.0190	0.0133	0.0319	0.0641
Translation: After	0.0005	0.0005	0.0014	0.0019
Rotation: Before	3.3085×10^{-06}	2.2029×10^{-06}	7.216×10^{-06}	1.1024×10^{-05}
Rotation: After	1.4779×10^{-06}	9.3516×10^{-07}	3.7528×10^{-06}	4.8574×10^{-06}
Scale: Before	0.0025	0.0001	0.0027	0.0070
Scale: After	1.7478×10^{-05}	4.4299×10^{-06}	2.3783×10^{-05}	5.0806×10^{-05}
Unconstrained: Before	0.0004	0.0002	0.0007	0.0012
Unconstrained: After	2.4401×10^{-05}	4.7291×10^{-05}	0.0001	0.0001

Table 99. Sampson Distance Optimization: Noisy Cube Projections.

Comparison of the errors in the epipolar geometry before and after the minimization of the Sampson Distance.

Motion	Average	Standard Deviation	Max	L2 Norm
Translation: Before	4.3668×10^{-16}	9.0820×10^{-16}	7.1054×10^{-16}	1.4187×10^{-16}
Translation: After	0.0035	0.0022	0.0101	0.0578
Rotation: Before	4.8537×10^{-16}	6.2528×10^{-16}	3.5527×10^{-15}	1.1149×10^{-14}
Rotation: After	1.1180×10^{-15}	6.4307×10^{-16}	2.6645×10^{-15}	1.8183×10^{-14}
Scale: Before	5.5966×10^{-16}	9.7282×10^{-16}	9.2328×10^{-15}	1.5803×10^{-14}
Scale: After	4.6669×10^{-15}	4.6937×10^{-15}	2.2919×10^{-14}	9.3254×10^{-14}
Unconstrained: Before	9.2570×10^{-16}	1.5917×10^{-15}	1.0658×10^{-14}	2.5926×10^{-14}
Unconstrained: After	2.2171×10^{-05}	2.1343×10^{-15}	1.3323×10^{-14}	4.3360×10^{-14}

Table 100. Sampson Distance Optimization: Noiseless Cloud Projections.

Comparison of the errors in the epipolar geometry before and after the minimization of the Sampson Distance.

Motion	Average	Standard Deviation	Max	L2 Norm
Translation: Before	0.0087	0.0122	0.0894	0.2122
Translation: After	0.0104	0.0116	0.0660	0.2192
Rotation: Before	0.0233	0.0181	0.0814	0.4160
Rotation: After	0.0096	0.0083	0.0382	0.1782
Scale: Before	0.0222	0.0212	0.1242	0.4326
Scale: After	0.0190	0.0278	0.2045	0.4744
Unconstrained: Before	0.0109	0.0132	0.0749	0.2415
Unconstrained: After	0.0378	0.0375	0.2069	0.7503

Table 101. Sampson Distance Optimization: Noisy Cloud Projections.

Comparison of the error in the epipolar geometry before and after the minimization of the Sampson Distance for the data set Cloud: Noisy Points.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D: REPROJECTION ERROR OPTIMIZATION RESULTS

Motion	Average	Standard Deviation	Max	L2 Norm
Translation: Before	0.0890	0.0852	0.2246	0.3379
Translation: After	66.7260	96.8058	226.8403	318.1485
Rotation: Before	3.9378×10^{-16}	2.4332×10^{-16}	8.8818×10^{-16}	1.2865×10^{-15}
Rotation: After	2.5058×10^{-08}	5.6382×10^{-08}	1.6391×10^{-07}	1.6515×10^{-07}
Scale: Before	2.5299×10^{-13}	4.9432×10^{-13}	1.4515×10^{-12}	1.4908×10^{-12}
Scale: After	5.5565×10^{-16}	5.1918×10^{-15}	8.5901×10^{-15}	1.3826×10^{-14}
Unconstrained: Before	0.7366	0.9854	2.8949	3.3374
Unconstrained: After	7.6233×10^{-08}	5.2909×10^{-08}	1.4351×10^{-07}	2.5708×10^{-07}

Table 102. Gold Standard Optimization: Noiseless Cube Projections.

Comparison of the errors in the epipolar geometry before and after optimization using the Gold Standard method.

Motion	Average	Standard Deviation	Max	L2 Norm
Translation: Before	0.0647	0.0530	0.1401	0.2306
Translation: After	5.1867	4.3887	14.4881	18.7094
Rotation: Before	0.0004	0.0002	0.0007	0.0110
Rotation: After	0.0030	0.0027	0.0076	0.0110
Scale: Before	0.0061	0.0038	0.0138	0.0201
Scale: After	0.0002	0.0043	0.0071	0.0113
Unconstrained: Before	0.0004	0.0001	0.0005	0.0013
Unconstrained: After	0.0003	0.0003	0.0010	0.0013

Table 103. Gold Standard Optimization: Noisy Cube Projections.

Comparison of the errors in the epipolar geometry before and after optimization using the Gold Standard method.

Motion	Average	Standard Deviation	Max	L2 Norm
Translation: Before	4.3668×10^{-16}	9.0820×10^{-16}	7.1054×10^{-15}	1.4187×10^{-14}
Translation: After	$1.2150 \times 10^{+15}$	$2.7645 \times 10^{+15}$	$2.2518 \times 10^{+16}$	$4.2509 \times 10^{+16}$
Rotation: Before	4.8537×10^{-16}	6.2528×10^{-16}	3.5527×10^{-15}	1.1149×10^{-14}
Rotation: After	1.1243×10^{-12}	3.6434×10^{-13}	2.2737×10^{-12}	1.6668×10^{-11}
Scale: Before	5.5966×10^{-16}	9.7282×10^{-16}	9.2328×10^{-15}	1.5802×10^{-14}
Scale: After	3.4280×10^{-16}	1.0661×10^{-15}	9.2325×10^{-15}	1.5801×10^{-14}
Unconstrained: Before	9.2570×10^{-16}	1.5917×10^{-15}	1.0658×10^{-14}	2.5926×10^{-14}
Unconstrained: After	2.8040×10^{-13}	4.1222×10^{-13}	2.1601×10^{-12}	7.0208×10^{-12}

Table 104. Gold Standard Optimization: Noiseless Cloud Projections.

Comparison of the errors in the epipolar geometry before and after optimization using the Gold Standard method.

Motion	Average	Standard Deviation	Max	L2 Norm
Translation: Before	0.0290	0.0385	0.2386	0.6782
Translation: After	129.8705	177.1689	1093.8964	3093.7659
Rotation: Before	0.0908	0.0746	0.3721	1.6563
Rotation: After	45.2342	37.2475	172.7441	825.7605
Scale: Before	0.0283	0.0303	0.2123	0.5843
Scale: After	0.0006	0.5668	2.2206	7.9830
Unconstrained: Before	0.0577	0.0447	0.2288	1.0290
Unconstrained: After	35.1666	26.8158	141.3355	623.2811

Table 105. Gold Standard Optimization: Noisy Cloud Projections.

Comparison of the errors in the epipolar geometry before and after optimization using the Gold Standard method.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision.* 2nd ed.Cambridge University Press, 2003.

[2] O. Faugeras and Q. Luon, *The Geometry of Multiple Images.* MIT Press, 2001.

[3] D. H. Kolb, D. E. Fernandez and D. R. Nelson. (2005, September, 2005). Webvision: The organization of the retina and visual system. *2007(August 2007).*

[4] Eastman Kodak Company, "Charge-Coupled Device (CCD) Image Sensors," vol. Rev 1, 2001.

[5] E. Trucco and A. Alessandro, *Introductory Techniques for 3-D Computer Vision.* 1st ed.Prentice Hall, 1998.

[6] J. Shi and C. Tomasi, "Good features to track," in *IEEE Conference on Computer Vision and Pattern Recognition,* 1994, pp. 593-600.

[7] Intel. (2006, Open source computer vision library (OpenCV). *1.0* Available: http://sourceforge.net/projects/opencvlibrary/.

[8] J. Bouguet. (2002, Pramidal implementation of the lucas-kanade feature tracker. *OpenCV Documentation* Available: http://www.intel.com/technology/computing/opencv/index.htm. *2007 (08/03).*

[9] P. J. Rousseeuw and A. M. Leroy, *Robust Regression and Outlier Detection.* New York, NY: John Wiley and Sons, Inc., 1987.

[10] P. D. Sampson. (1982, Fitting conic section to 'very scattered' data: An iterative refinement of the bookstein algorithm. *Computer Vision, Graphics, and Image Processing 18.* pp. 97-108.

[11] F. L. Bookstein, "Fitting Conic Sections to Scattered Data," *Computer Graphics and Image Processing,* vol. 9, pp. 56-71, 1979.

[12] Y. Ma, S. Soatto, J. Kosecka and S. S. Sastrym, *An Invitation to 3-D Vision.* New York, NY: Springer, 2006.

[13] D. J. Felleman and D. C. van Essen, "Distributed hierarchical processing in the primate cerebral cortex." in *Cerebral Cortex* , vol. 10, A. Peters and K. Rockland, Eds. 1991, pp. 1-47.

[14] E. Grayzeck and E. V. Bell. NASA IMAGE science center: Imager for magentopause-to-aurora global exploration. *2007(Aug 2007).*

[15] M. Irani and P. Anandan, "Parallax geometry of pairs of points for 3D scene analysis." in *European Conference on Computer Vision,* 1996, pp. 17-30.

[16] A. J. Davison, I. D. Reid, N. D. Molton and O. Stasse, "MonoSLAM: Real-Time Single Camera SLAM," *In IEEE Trans. PAMI,* 2007.

[17] S. Thrun, W. Burgard and D. Fox, "FastSLAM algorithm," in *Probabilistic Robotics* Anonymous Cambridge, Massachusetts: Massachusetts Institute of Technology Press, 2005, pp. 437-483.

[18] E. Eade and T. Drummond, "Scalable monocular SLAM," in *Computer Vision and Pattern Recognition (CVPR),* 2006, pp. 469-476.

[19] M. Tomono, "3-D localization and mapping using a single camera based on structure-from-motion with automatic baseline selection." in *International Conference on Robotics and Automation,* 2005, pp. 3342-3347.

[20] X. Armangue and J. Salvi, "Overall View Regarding Fundamental Matrix Estimation," *Image and Vision Computing,* pp. 205-220, 2003.

[21] S. Knorr, A. Smolic and T. Sikora, "From 2D- to stereo- to multi-view video," in *3DTV-Con,* 2007.

[22] M. Pollefeys, L. Van Gool, M. Vergauwen, F. Verbiest, K. Cornelis and J. Tops, "Visual Modeling with a Hand-Held Camera," *International Journal of Computer Vision,* vol. 59, pp. 207-232, 2004.

[23] T. Jebara, A. Azarbayejani and A. Pentland, "3D Structure From 2D Motion: The Inverse Hollywood Problem: Getting Models and Motion Out Of Video For Post-Production, MPEG, and More," *IEEE Signal Processing Magazine,* pp. 66-84, May 1999. 1999.

[24] A. Saxena, "Learning Depth from Single Monocular Images," 2006.

[25] L. Guan, J. Franco and M. Pollefeys, "3D occlusion inference from silhouette cues," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2007, pp. 1-8.

[26] H. C. Longuet-Higgins, "A computer algorithm for reconstructing a scene from two projections." *Nature,* vol. 293, pp. 133-135, 10 September 1981. 1981.

[27] P. H. S. Torr. (2004, Structure and motion toolkit.) Available: http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=4576&objectType=File.

[28] P. H. S. Torr, "Motion Segmentation and Outlier Detection," 1995.

[29] P. H. S. Torr and A. Zisserman, "MLESAC: A New Robust Estimator with Application to Estimating Image Geometry," *Computer Vision and Image Understanding,* vol. 78, pp. 138-156, 2000.

[30] K. Strobl, W. Sepp, S. Fuchs, C. Paredes and K. Arbter. (2004, Camera calibration toolbox for matlab.) Available: http://www.vision.caltech.edu/bouguetj/calib_doc/. *2007 (05/12).*

[31] MathPages.com. Pythagoras on dot and cross products. *2007(06/08).*

[32] P. Sinha, I. Blthoff and H. Blthoff, " Top-Down Influences On Stereoscopic Depth-Perception." *Nature Neuroscience,* pp. 254-257, 1998.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Fort Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Marine Corps Representative
   Naval Postgraduate School
   Monterey, California

4. Director, Training and Education, MCCDC, Code C46
   Quantico, Virginia

5. Director, Marine Corps Research Center, MCCDC, Code C40RC
   Quantico, Virginia

6. Marine Corps Tactical Systems Support Activity (Attn: Operations Officer)
   Camp Pendleton, California